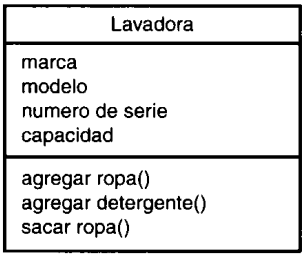
Hora 1 - Introducción a UML

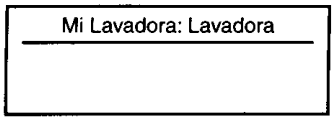
La finalidad de los diagramas es presentar diversas perspectivas de un sistema, las cuales se conocen como modelo. UML describe lo que hará el sistema, pero no cómo lo hará.

- **Diagrama de clases:** las cosas que nos rodean tienen atributos y realizan acciones y están categorizadas. Estas categorías se llaman clases. Una clase es una categoría o un grupo de cosas que tienen atributos y acciones similares.

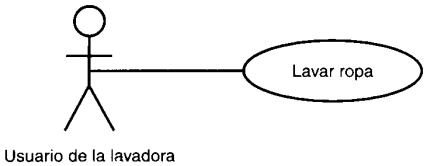


  Los diagramas de clases facilitan las representaciones a partir de las cuales los desarrolladores podrán trabajar. A su vez, colaboran con el análisis, permitiendo hablar con los clientes en su propia terminología, lo que hace que ellos puedan detallar mejor los problemas a resolver.

- **Diagrama de objetos:** un objeto es una instancia de una clase, tiene valores específicos para los atributos y las acciones. Se diferencia de la clase en que el nombre se subraya. El nombre de la instancia se encuentra a la izquierda de los dos puntos (:) y el nombre de la clase a la derecha.

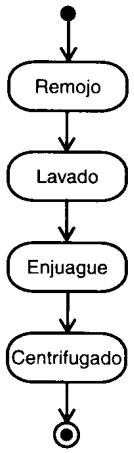


- **Diagrama de casos de uso:** un caso de uso es una descripción de las acciones de un sistema desde el punto de vista del usuario. Permite obtener los requerimientos del sistema desde el punto de vista del usuario.

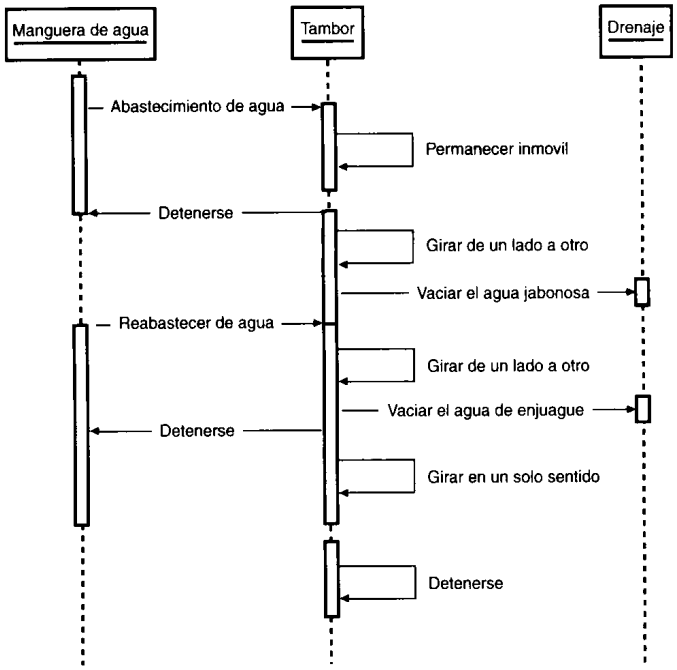


  A la figura correspondiente al usuario se lo conoce como actor. La elipse representa el caso de uso. El actor, que es quien inicia el caso de uso, puede ser una persona o un sistema.

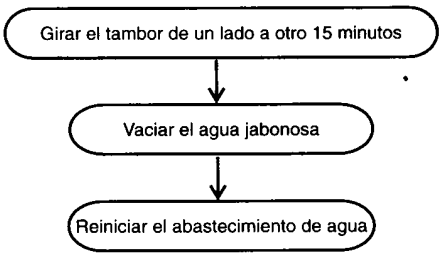
- **Diagrama de estados:** en cualquier momento, un objeto se encuentra en un estado particular. El símbolo que está en la parte superior de la figura representa el estado inicial y el de la parte inferior el estado final.



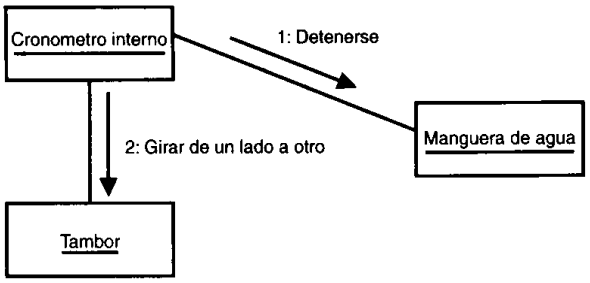
- **Diagrama de secuencias:** los diagramas de clases y objetos presentan información estática. El diagrama de secuencia muestra la mecánica de la interacción con base en el tiempo. El diagrama de secuencia captura las interacciones que se realizan a través del tiempo.



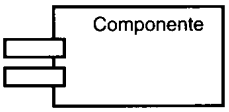
- **Diagrama de actividades:** las actividades que ocurren dentro de un caso de uso o dentro del comportamiento de un objeto se dan secuencialmente.



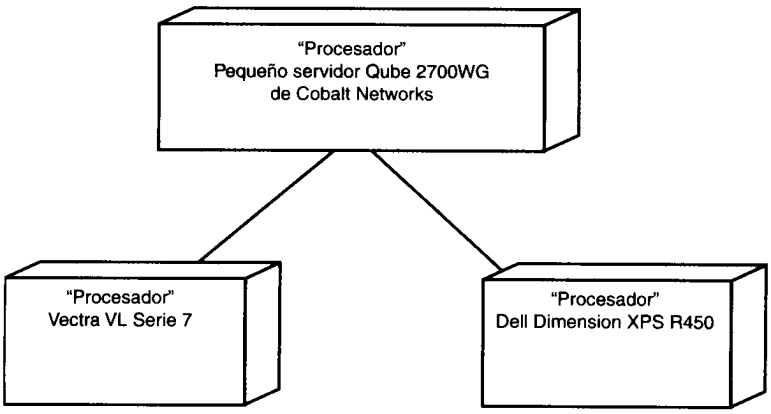
- **Diagrama de colaboraciones:** los elementos de un sistema trabajan en conjunto para cumplir con los objetivos.



- **Diagrama de componentes:** está íntimamente ligado con el mundo de los sistemas informáticos. El desarrollo de software se realiza mediante componentes, lo que es importante en los procesos de desarrollo en equipo.

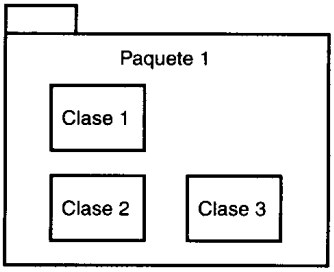


- **Diagramas de distribución:** muestra la arquitectura física de un sistema informático, representando los equipos, dispositivos, interconexiones y el software que hay en cada máquina.

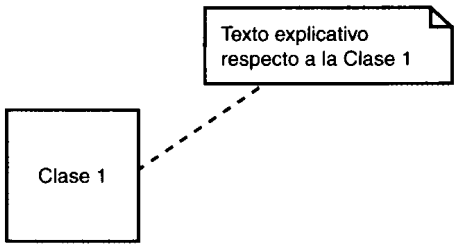


Otras características:

- **Paquetes:** permiten mostrar que ciertas clases o componentes son parte de un subsistema en particular.



- **Notas:** son el equivalente gráfico de un papel adherente. Permiten explicar mejor alguna parte del diagrama.



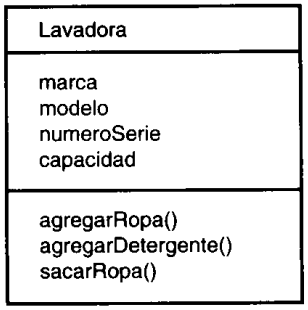
- **Estereotipos:** permiten tomar elementos propios del UML y convertirlos en otros. Se representan con un nombre entre dos pares de mayor y menor (<< >>) y después se los aplica correctamente. Ejemplo: interfaz. La interfaz es una clase que tiene operaciones pero no atributos. Son acciones que seguro se usan mucho en el modelo. Entonces, en vez de inventar un nuevo elemento para la interfaz, se usa el símbolo de una clase con <<Interfaz>> situada justo sobre el nombre de la clase.

La diversidad de diagramas nos permite tener variados y amplios puntos de vistas y modelos del sistema, lo que nos beneficia al tener muchas más perspectivas y poder abarcar mejor el problema. Los sistemas pueden estar modelados con un subconjunto de los diagramas, no hace falta que estén todos.

Hora 2 - Orientación a Objetos

Es una metodología basada en componentes donde se genera un sistema mediante un conjunto de objetos, el cual luego podremos ampliar y modificar, agregando nuevos componentes. Esto nos permite reutilizar los objetos y reducir sustancialmente el tiempo de desarrollo.

Objeto: es una instancia de una clase. Tiene una estructura formada por atributos y acciones. Las acciones son actividades que el objeto realiza. Las clases sirven como plantilla para fabricar objetos.

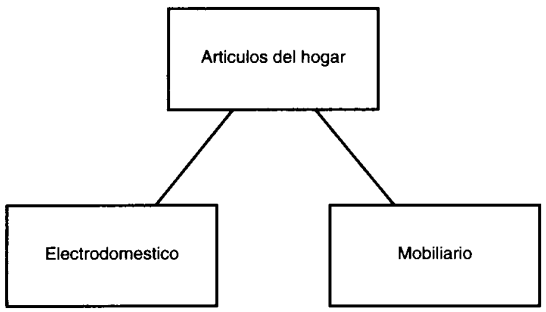


El propósito de la orientación a objetos es desarrollar software que refleje particularmente (que modele) un esquema del mundo.

Algunos conceptos:

- **Abstracción:** es quitar las propiedades y acciones de un objeto para dejar solo aquellas que son necesarias. Dependiendo del tipo de problema, requeriremos más o menos información. En cualquier caso, lo que tendremos luego de tomar la decisión de lo que se incluirá o desechará, será una abstracción del problema.

- **Herencia:** Como instancia de una clase, un objeto tiene todas las características de la clase que proviene. Caja objeto hereda los atributos y operaciones de dicha clase. Un objeto no solo hereda de una clase, sino que una clase también puede heredar de otra. Esto genera jerarquías de clases con subclases y superclases. Cada subclase puede ser superclase también y viceversa.



- **Polimorfismo:** Se refiere a la posibilidad de acceder a un variado rango de funciones distintas a través del mismo interfaz. Es decir, que una operación tenga el mismo nombre en diferentes clases y en cada caso hará algo diferente. Cada clase "sabe" cómo realizar dicha operación. También tenemos la sobrecarga que implica que una

misma función (mismo nombre) haga distintas cosas según los argumentos que se le pasen o según el objeto que la llame.

El polimorfismo ayuda no sólo al programador, sino también al modelador, quien podrá comunicarse con el cliente en sus propias palabras, manteniendo una terminología sin tener que crear palabras artificiales para sustentar una unicidad innecesaria en los términos.

- **Encapsulamiento:** se refiere a la capacidad de agrupar y condensar en un entorno con límites bien definidos distintos elementos. Es decir, cada objeto encapsula lo que hace, la funcionalidad interna. Esto permite reducir los errores potenciales, ya que cuando un objeto falla y hay que modificarlo, el ocultamiento podría evitarnos tener que modificar otros objetos. Pese a esto, todos los objetos deben presentar interfaces al mundo exterior para poder interactuar. Esto se logra con el envío de mensajes, que es cuando un objeto envía a otro un mensaje para realizar una operación y el receptor la ejecuta; o con las asociaciones, que son distintos tipos de relaciones que tienen entre sí los objetos. Estas pueden ser:

    - Unidireccionales o bidireccionales, dependiendo de qué tipo de relación tengan los objetos. Una clase se puede asociar con más de una clase distinta.
    - La multiplicidad es importante, ya que indica la cantidad de objetos de una clase que se relacionan con otro objeto de la clase asociada.

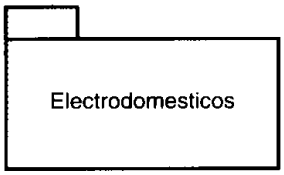
- **Agregación:** Es un tipo de asociación donde un objeto está constituido por diversos tipos de componentes.

- **Composición:** Es un tipo de agregación especial, donde el componente se considera como tal sólo como parte del objeto compuesto. Un objeto compuesto puede no tener el mismo tiempo de vida que sus componentes.

Hora 3 - Uso de la orientación a objetos

Las clases se representan con un rectángulo. El nombre de la misma se escribe con una palabra con la primera letra en mayúscula y se coloca en la parte superior del rectángulo. Si el nombre de la clase tiene 2 palabras, se unen y se inicia cada una con mayúscula.

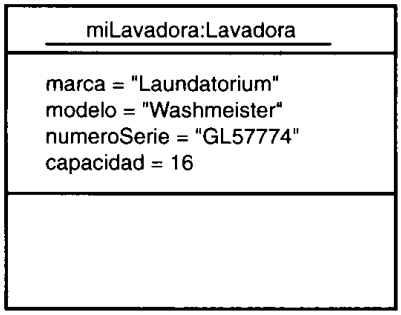
El paquete puede jugar un papel en el nombre de la clase.



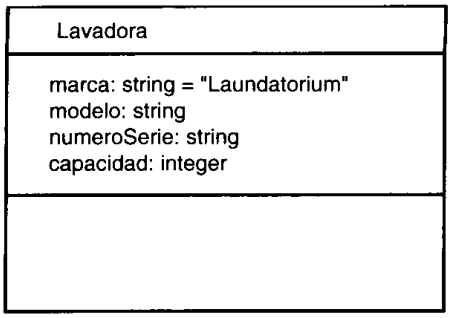
El par de dos puntos (:) separa al nombre del paquete, que está a la izquierda, del nombre de la clase que va a la derecha.

Atributos: Son propiedades de una clase y describen un rango de valores que dicha propiedad puede contener en los objetos. Una clase puede tener varios o ningún atributo. Si el nombre del atributo es una sola palabra, se indica en minúscula. Si tiene más de una, cada palabra se une a la anterior y comienza con una mayúscula, a excepción de la primera palabra.

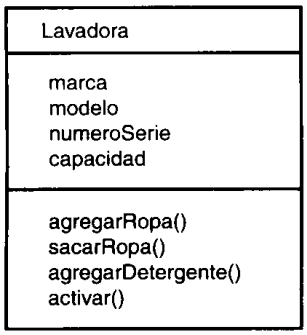
Todo objeto de la clase tiene un valor específico en cada atributo. El nombre de un objeto se inicia con una letra minúscula y está precedido de dos puntos que a su vez están precedidos del nombre de la clase y todo el nombre está subrayado.



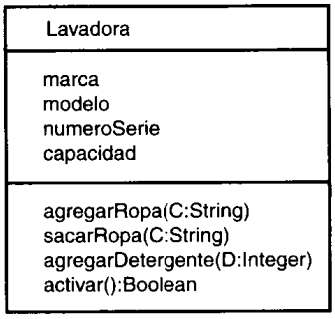
UML da la opción de indicar información adicional de los atributos. Se puede especificar un tipo para cada valor del atributo, usando los dos puntos para separar el nombre del atributo de su tipo. También se puede indicar un valor predeterminado para el atributo.



Operaciones: Es algo que la clase puede realizar o que un usuario o sistema puede hacer a una clase. Su nombre se escribe en minúscula si tiene una sola palabra. Si tiene más, se unen y se inician todas con mayúscula excepto la primera.

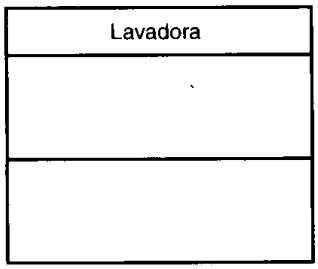


En los paréntesis que preceden al nombre de la operación, se pueden mostrar los parámetros con los que funciona, así como el tipo de dato. También se puede mostrar el tipo de valor que regresa.

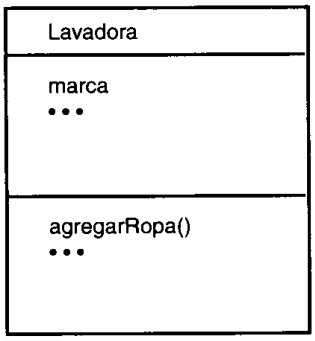


Atributos, operaciones y concepción

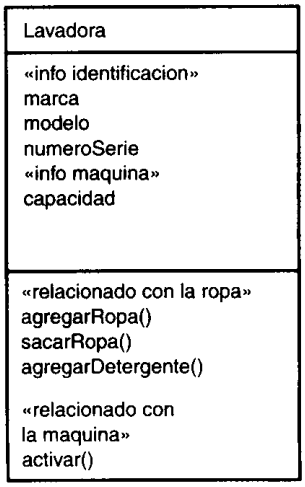
Cuando se diagrama, se coloca más de una clase a la vez. Por ello, no siempre es útil que se vean todos los atributos y operaciones, lo cual haría muy saturado al diagrama. Por lo tanto podemos mostrar sólo el nombre de la clase y dejar vacías las áreas de atributos u operaciones.



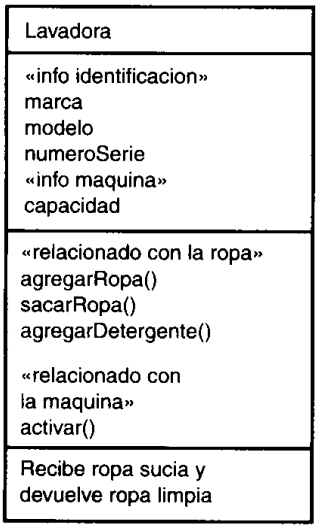
También pueden mostrarse solo algunos atributos u operaciones, indicando luego de esa lista tres puntos suspensivos (…), lo que nos genera una clase abreviada.



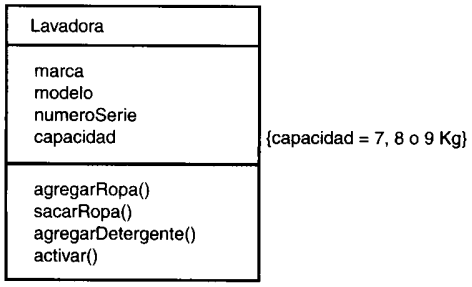
Si la lista de atributos y operaciones es larga, se pueden usar estereotipos para organizarla de manera que sea más comprensible. Esto se indica con el nombre bordeado por dos pares de mayor y menor (<< >>).



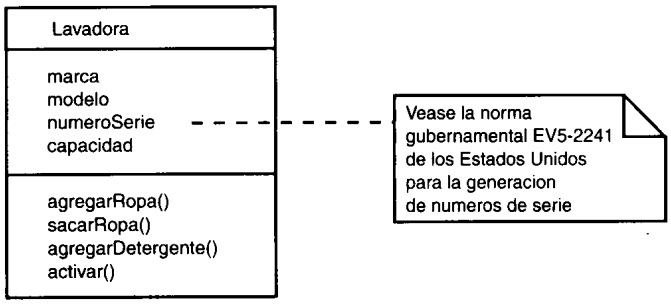
Responsabilidades y restricciones: Podemos agregar otra información en cada clase. Debajo de las operaciones, podemos mostrar la *responsabilidad* de la clase, es decir, una descripción de lo que hace, lo que sus atributos y operaciones realizan en conjunto.



La idea de esto es incluir información suficiente para describir una clase de manera inequívoca. También existen las restricciones, que especifican una o varias reglas que sigue la clase. Se indican con un texto bordeado por llaves.



Notas adjuntas: Por encima y debajo de los atributos, operaciones, responsabilidades y restricciones, puede agregar mayor información a una clase con las notas adjuntas.



Estas notas pueden contener imágenes y texto

Qué hacen las clases y cómo encontrarlas

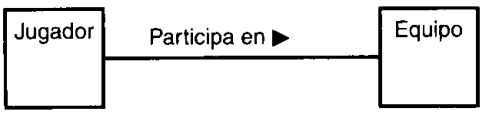
Las clases son el vocabulario y terminología de un área del conocimiento.

En las reuniones con los clientes, prestar atención a los sustantivos que usan para describir las entidades. También a los verbos, que se convertirán en operaciones. Los atributos surgen como sustantivos relacionados con los nombres de las clases. También debemos preguntar qué es lo que cada clase hace dentro del negocio. Estas respuestas nos indicarán la responsabilidad de cada clase.

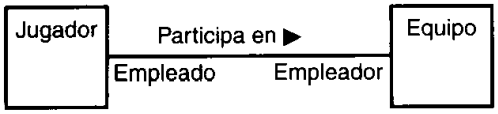
Hora 4 - Uso de relaciones

Las relaciones son las conexiones entre las clases y su representación.

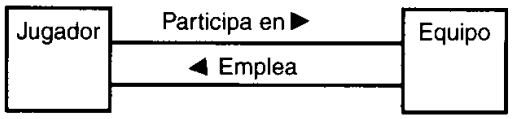
Asociaciones: es cuando las clases se conectan entre sí de forma conceptual. Se representa con una línea que conecta ambas clases con el nombre de la asociación sobre la línea. Se debe indicar la dirección de la relación con un triángulo relleno que apunte a la dirección apropiada.



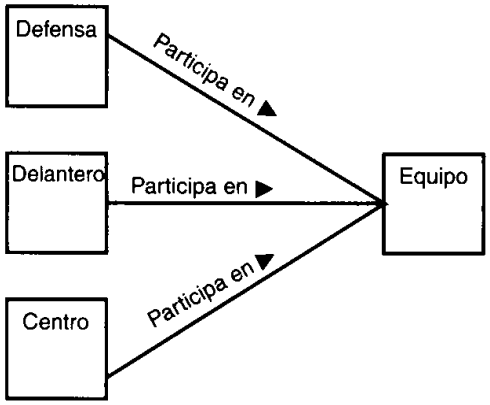
Cuando una clase se asocia con otra, cada una de ellas juega un papel dentro de la asociación. Estos papeles se representan escribiéndolos cerca de la línea que se encuentra junto a la clase que juega el papel correspondiente.



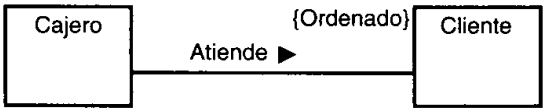
La asociación puede funcionar en dirección inversa. Esto se representa indicando ambas asociaciones en el diagrama con un triángulo relleno en cada dirección.



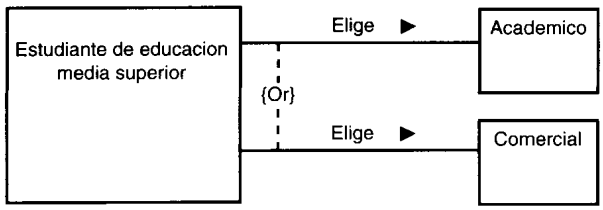
También varias clases se pueden conectar a una.



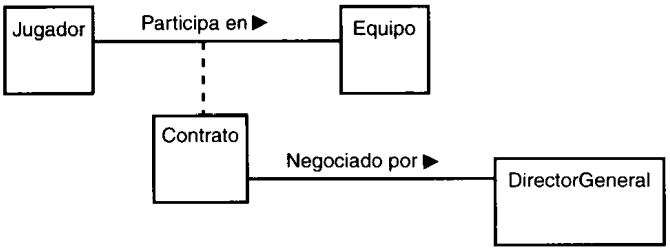
Restricciones en las asociaciones: se indican junto a la línea de asociación colocando la palabra entre llaves.



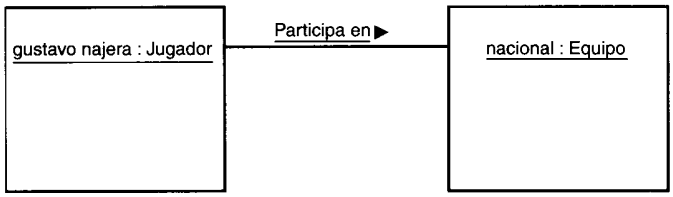
Otro tipo de restricción es la relación O (Or) en una línea discontinua que conecta a dos líneas de asociación.



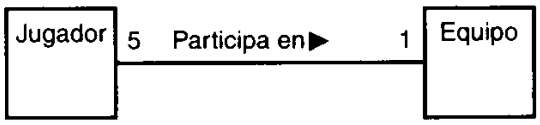
Clases de asociación: una asociación puede tener atributos y operaciones, de manera de ser tratada como una clase estándar. Se usa una línea discontinua para conectarla a la línea de asociación. Una clase de asociación puede tener asociaciones con otras clases.



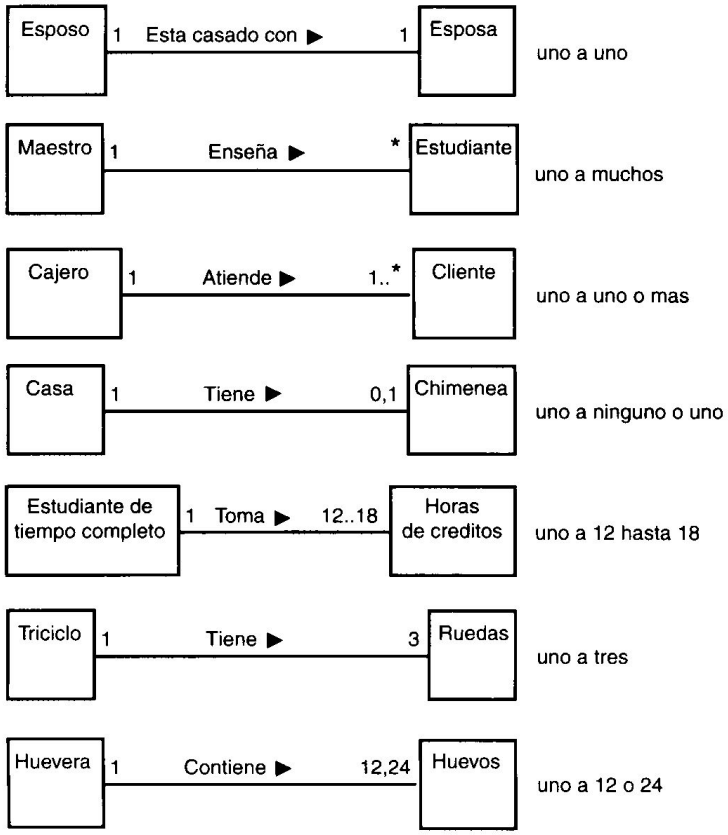
Vínculos: son instancias de una asociación. Conecta dos objetos en lugar de dos clases. Se debe subrayar el nombre del vínculo, como se hace en el nombre de un objeto.



Multiplicidad: Es la cantidad de objetos de una clase que se relacionan con un objeto de la clase asociada. Se colocan los números sobre la línea de asociación junto a la clase correspondiente.



Tipos de multiplicidad:

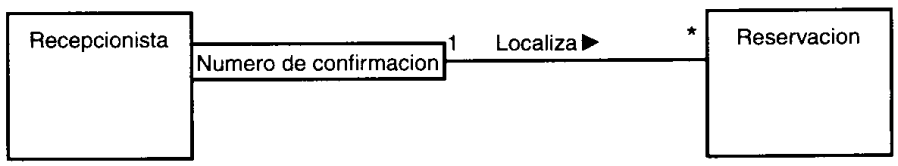


UML usa un asterisco para representar *más* y para representar *muchos*. En un contexto O (Or) se representa por dos puntos como en 1..* (uno o más). En otro contexto, O (Or) se representa por una coma, como en 5,10 (cinco o diez).

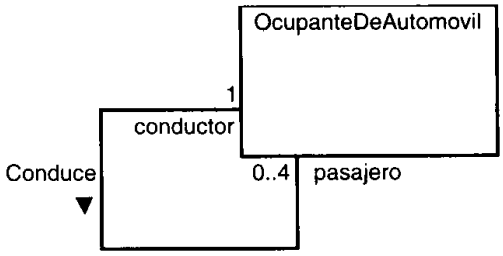
Cuando la clase A tiene una multiplicidad de uno a ninguno o uno con la clase B, se dice que B es opcional para la clase A.

Asociaciones calificadas: cuando un objeto de una clase tiene que buscar un objeto en particular de otro tipo para cumplir con un papel en la asociación, la primera clase debe atenerse a un atributo en particular para localizar al objeto adecuado. Este atributo suele es un identificador que puede ser un nº de identidad.

Esta información de identidad se llama *calificador*. El símbolo es un pequeño rectángulo adjunto a la clase que hará la búsqueda.



Asociaciones reflexivas: asociación de una clase consigo misma. Esto puede ocurrir cuando una clase tiene objetos que pueden jugar diversos papeles. Se representa mediante una línea de asociación que sale y llega a la misma clase y sobre la línea indicamos papeles, nombre, dirección y multiplicidad.



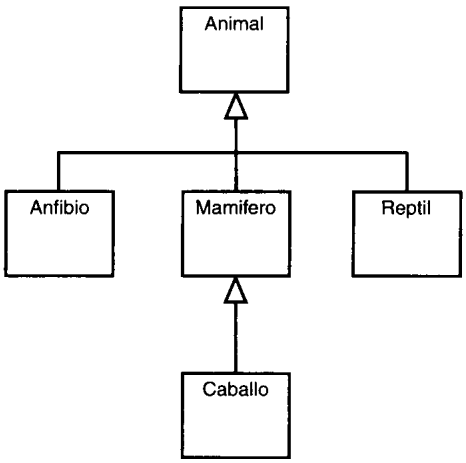
Herencia y Generalización: Surge del conocimiento que tenemos de una categoría de cosas, de donde deducimos algunas cosas que se pueden transferir a otras categorías.

Herencia o generalización es cuando una clase (secundaria o subclase) hereda los atributos y operaciones de otra clase (principal o superclase). La clase principal es más genérica que la secundaria.

Donde quiera que hagamos referencia a la superclase, también referenciamos a la subclase, pero no viceversa.

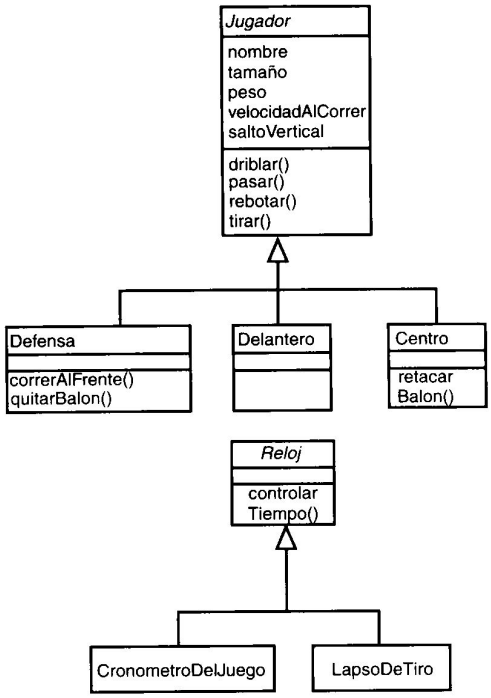
Una subclase puede ser superclase de otra subclase. En UML la herencia se representa con una línea que conecte la superclase con la/las subclases. En la parte de la línea que se conecta con la clase principal, se coloca un triángulo sin rellenar que apunte a la misma.

La relación que se aplica para la herencia es "es un tipo de".



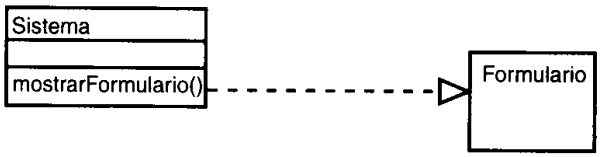
Descubrimiento de la herencia: Las clases candidatas que descubramos con los clientes pueden incluir tanto las superclases como las subclases. Debemos darnos cuenta que los atributos y operaciones de una clase son generales y se pueden aplicar a varias clases. También podemos notar que dos o más clases tienen ciertos atributos y operaciones en común.

Clases abstractas: Son clases que no proporcionan ninguna instancia al modelo. No proveen objetos. Se representan con el nombre en *cursiva*.



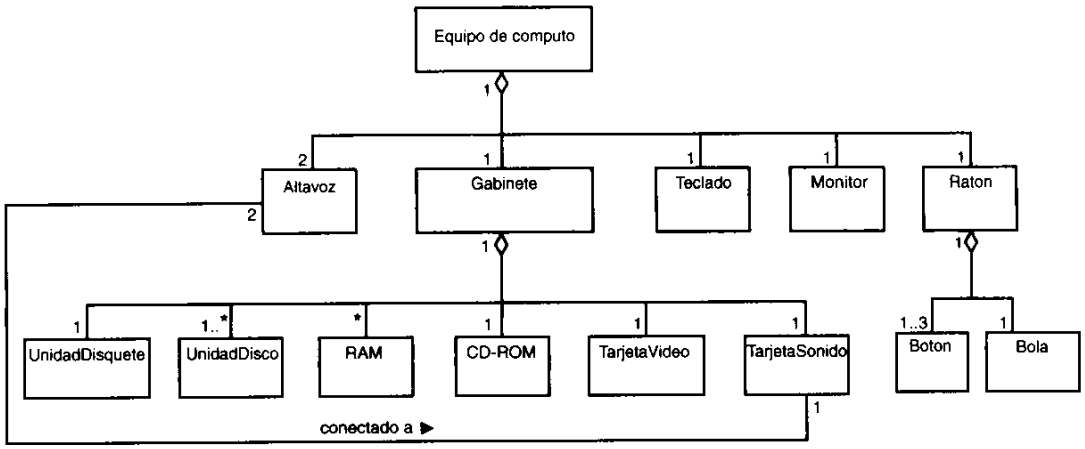
Dependencias: es cuando una clase usa a otra. El uso más común es mostrar que la firma de la operación de una clase utiliza a otra clase.

Se representa con una línea discontinua con una punta de flecha en forma de triángulo sin relleno que apunta a la clase de la que depende.

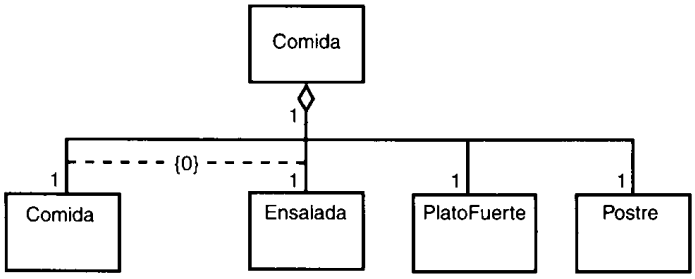


Hora 5 - Agregación, composición, interfaces y realización

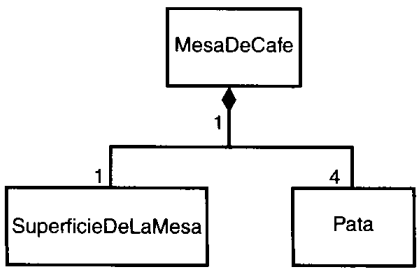
Agregación: es cuando una clase consta de otras clases. Los componentes y la clase que constituyen son una asociación que conforma un todo. Se representa como una jerarquía dentro de la clase completa en la parte superior y los componentes por debajo de ella. Una línea conecta el todo con un componente mediante un rombo sin relleno que se coloca en la línea más cercana al todo.



Restricciones: Se puede establecer esta relación dentro de una relación O (Or). Se indica la palabra O (Or) dentro de llaves con una línea discontinua que conecte las dos líneas que conforman al todo.

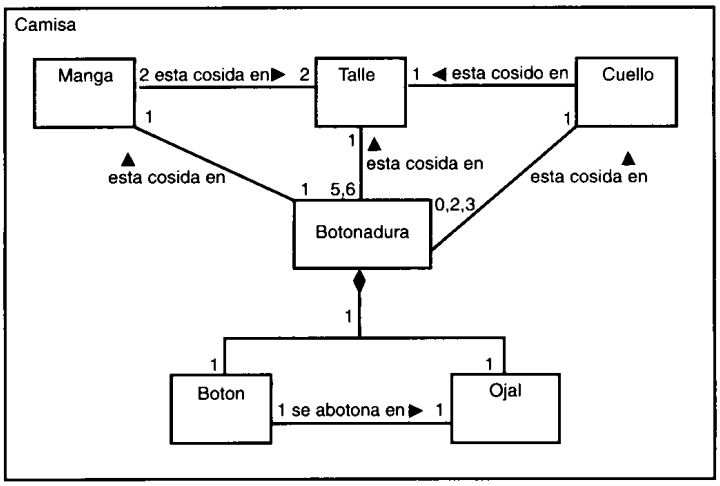


Composición: es un tipo muy representativo de agregación, donde cada componente dentro de una composición sólo puede pertenecer al todo. Se indica igual que la agregación, pero con el rombo relleno.

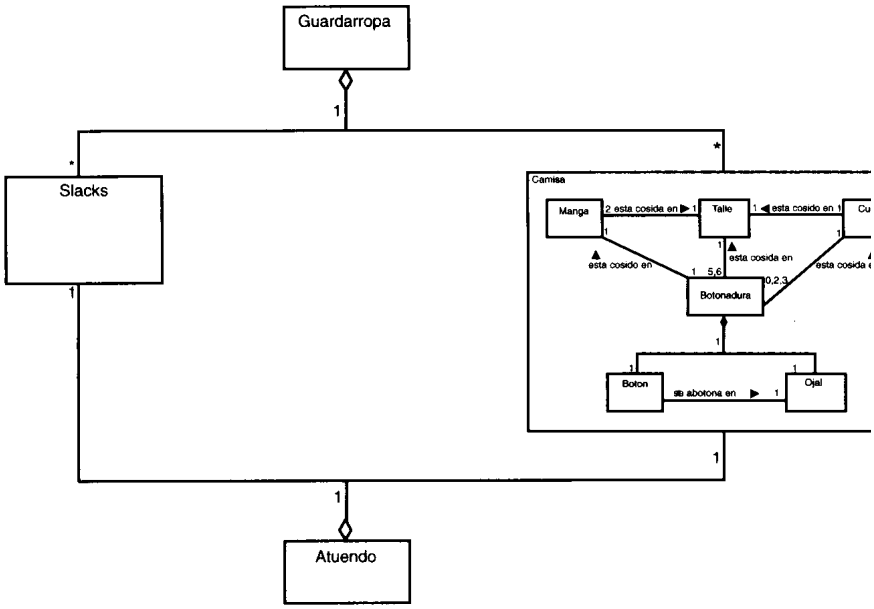


Contexto: cuando modelamos sistemas y surgen las relaciones anteriores, debemos enfocar la atención en qué grupo es y el diagrama de contexto nos dará las características del modelo que se requiera para tal fin. Las composiciones figuran en gran medida dentro de los diagramas de contexto. Este diagrama es como un mapa detallado de alguna sección de un mapa mayor. Pueden necesitarse varias secciones para capturar la información detallada.

Este diagrama muestra cómo los componentes están relacionados entre sí. Un diagrama de contexto de composición muestra los componentes de una clase como un diagrama anidado dentro de un enorme rectángulo de clase.



Para mostrar el todo de la composición en el contexto mayor que lo abarca, hay que ampliar el ámbito. Un diagrama de contexto del sistema lo hará por nosotros. Se muestra la forma en que la clase compuesta se conecta con las otras clases.

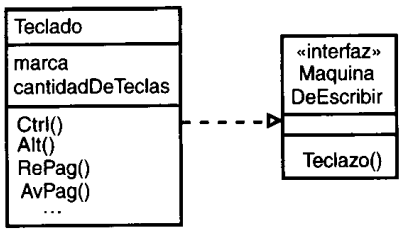


Se puede ver de cerca alguna otra clase y presentar sus detalles en otro diagrama de contexto.

Interfaces y realizaciones: Cuando deseemos contar con algún medio para capturar todas las operaciones reutilizables del modelo, debemos usar una interfaz. La interfaz es la estructura de UML que nos permite hacer esto. Una interfaz es un conjunto de operaciones que especifica cierto aspecto de la funcionalidad de una clase y es un conjunto de operaciones que una clase presenta a otras.

La interfaz puede establecer un subconjunto de las operaciones de una clase y no necesariamente todas ellas. Podemos modelar la interfaz igual que una clase, con un símbolo rectangular. La diferencia es que la interfaz no tiene atributos. Las maneras de distinguir una interfaz de una clase que no muestra sus atributos son usando un estereotipo y especificando la palabra <<interfaz>> en el rectángulo, o colocar la letra 'I' al principio del nombre de una interfaz.

La relación entre una clase y una interfaz se conoce como *realización*. Esta relación se representa con una línea discontinua con una punta de flecha en forma de triángulo sin rellenar que adjunte y apunte a la interfaz.



Una clase puede realizar más de una interfaz y una interfaz puede ser realizada por más de una clase.

Visibilidad: se aplica a atributos y operaciones de una clase y define en qué medida otras clases pueden usarlos (también aplicado a operaciones de interfaz) Hay tres niveles:

- Público: la funcionalidad se extiende a otras clases. Usa el símbolo **+.**
- Protegido: la funcionalidad se otorga solo a las clases que heredan de la clase original. Usa el símbolo **#.**
- Privado: solo la clase original puede usar la funcionalidad. Usa el símbolo **-.**

La realización implica que el nivel público se aplique a cualquier operación en una interfaz.

Ámbito: hay dos tipos:

- De instancia: cada instancia cuenta con su propio valor en un atributo u operación. Es el ámbito más común.
- Archivado: solo hay un valor del atributo u operación en todas las instancias de clase. Aparece con el nombre subrayado. Se usa cuando un grupo específico de instancias tienen que compartir los valores de un atributo privado.

Hora 6 - Introducción a los casos de uso

Es importante la visión del usuario, comprender su punto de vista es clave para generar sistemas que sean útiles y funcionales. Esto implica que cumplan con los requerimientos y que sea fácil trabajar con ellos.

El modelado del sistema desde la perspectiva del usuario es el trabajo de los casos de uso.

Qué son los casos de uso

El análisis del caso de uso permite preguntar cómo se usará el producto o sistema que se va a comprar, de manera de que se obtenga algo que cumpla con las necesidades. Lo importante es saber cuáles son los requerimientos. La forma en que los usuarios usan el sistema nos da la pauta para diseñar y crear.

El caso de uso es una estructura que ayuda al analista a trabajar con los usuarios para determinar la forma en que se usará un sistema.

Caso de uso: es una colección de situaciones respecto al uso de un sistema. Cada situación describe una secuencia de eventos que se inician por una persona, sistema, parte de hardware o por el paso del tiempo. Las entidades que inician las secuencias se llaman *actores*. El resultado de la secuencia es algo utilizable, tanto por el actor que la inició o por otro.

Importancia de los casos de uso.

Sirven para estimular a que los usuarios hablen de un sistema desde su punto de vista. También involucran a los usuarios en las etapas iniciales del análisis y diseño del sistema. Esto aumenta la probabilidad de que el sistema sea de mayor provecho para la gente a la que ayudará.

Inclusión de los casos de uso

A veces es común distinguir pasos en común en las distintas secuencias. Podemos eliminar esta duplicación, tomando cada secuencia común y conformando un caso de uso adicional a partir de ellos.

Con estos nuevos casos de uso, los casos de los que derivan los incluyen dentro de la secuencia.

Extensión de los casos de uso

Es posible reutilizar un caso de uso de manera distinta a la inclusión. Podemos crear un caso de uso agregándole algunos pasos a un caso de uso existente. Este nuevo caso de uso es una extensión del original.

Inicio del análisis de un caso de uso

1. Entrevistas a los clientes para armar los diagramas de clase. Esto nos dará cierta idea del área de trabajo y una familiaridad con los términos que usaremos.
2. Entrevistas a los usuarios para que nos indiquen qué es lo que ellos harían con el sistema. Sus respuestas conformaran el conjunto candidato de casos de uso.
3. Describimos brevemente cada caso y derivamos una lista de todos los actores que iniciarán y se beneficiar

Los casos de uso aparecerán en varias fases del desarrollo y nos ayudarán con el diseño de una interfaz de usuario, con la programación y establecerán las bases para probar el sistema generado.

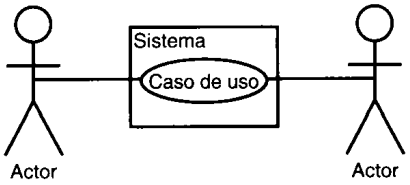
Hora 7 - Diagramas de casos de uso

Una de las finalidades del proceso de análisis es generar una colección de casos de uso. La idea es poder catalogar y referenciar dicha colección, que es el punto de vista del usuario acerca del sistema.

Representación de un modelo de caso de uso

Un actor inicia el caso de uso y otro (puede ser el mismo) recibirá algo de valor de él. Se representa con una elipse al caso de uso y una figura agregada al actor. El que inicia se indica a la izquierda del caso de uso y el que recibe a la derecha. El nombre del actor aparece debajo de la figura y el del caso de uso adentro de la elipse o justo abajo. Una línea asociativa conecta al actor con el caso de uso y representa la comunicación entre ambos.

Beneficios: los casos de uso muestran los confines del sistema. Generalmente los actores están fuera del sistema y los casos de uso adentro. Se usará un rectángulo con el nombre del sistema para representar los confines del mismo. El rectángulo envuelve los casos de uso.



Secuencia de pasos en los escenarios

Cada caso de uso es una colección de escenarios y cada escenario una secuencia de pasos. Cada diagrama tendrá su propia página, de igual manera, cada escenario tendrá su página, donde se listará:

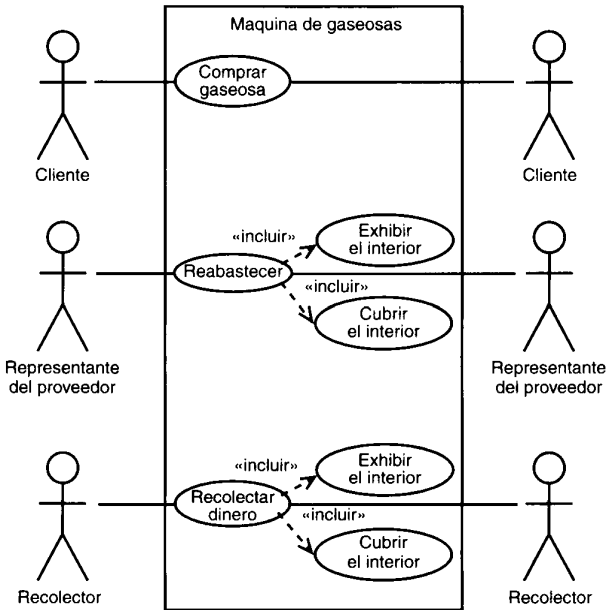
- El actor que inicia el caso de uso
- Condiciones previas para el caso de uso
- Pasos en el escenario
- Condiciones posteriores cuando se finaliza el escenario
- El actor que se beneficia del caso de uso

También podemos enumerar las conjeturas del escenario y una breve descripción de una sola frase sobre el mismo.

Escenarios alternativos: Podemos incluir los mismos de manera separada o considerarlos como excepciones al primer escenario del caso de uso.

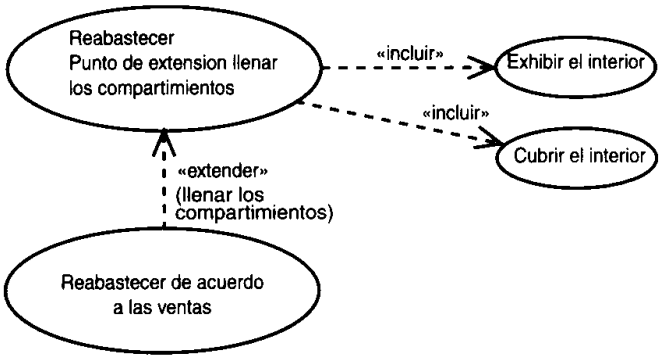
Concepción de las relaciones entre casos de uso

- Inclusión: volver a usar los pasos de un caso de uso dentro de otro. Se representa usando el símbolo que usamos para la dependencia entre clases: una línea discontinua con una punta de flecha que conecta los casos de uso apuntando al caso de uso incluido. Sobre la línea se agrega un estereotipo con la palabra <<incluir>>. En la notación de texto que sigue los pasos de la secuencia, indicamos los casos de uso incluidos.

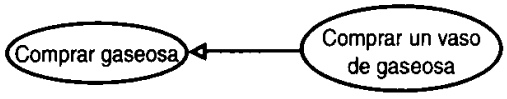


- Extensión: Crear un caso de uso adicionando pasos a uno existente, llamado caso de uso básico o base. La extensión solo se puede realizar en puntos indicados de manera específica en la secuencia. Estos puntos se llaman *puntos de extensión*.

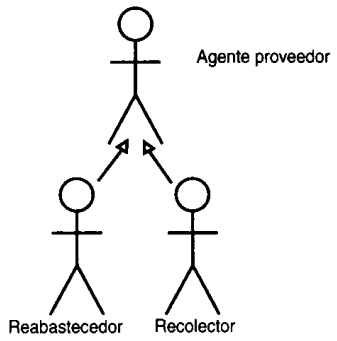
Se representa igual que la inclusión, pero con el estereotipo que muestra <<extender>>. Dentro del caso de uso básico, el punto de extensión aparece debajo del nombre del caso de uso.



- Generalización: Es cuando un caso de uso hereda las acciones y significado del primario y además agrega sus propias acciones. Se puede aplicar el caso de uso secundario en cualquier lugar donde apliquemos el primario. Se representa con una línea discontinua con una punta de flecha en forma de triángulo sin rellenar que apunta al caso de uso primario.



Esta relación puede establecerse entre actores también.



- Agrupamiento: Nos permite organizar y categorizar los casos de uso cuando tenemos, por ejemplo, muchos subsistemas dentro del sistema. La forma más directa de organizarlos es agrupar los casos de uso en un paquete.

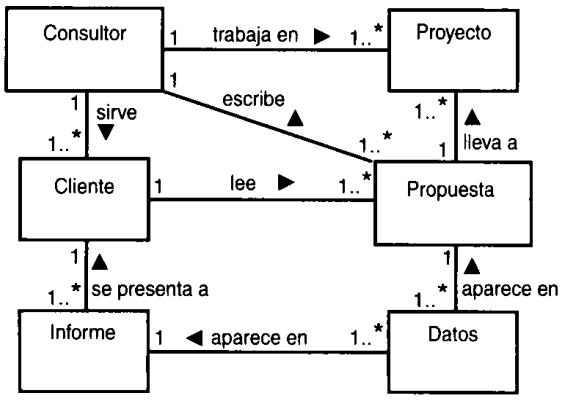
Diagramas de casos de uso en el proceso de análisis

Las entrevistas con los usuarios comienzan en la terminología del dominio, pero deben alternarse con la terminología del usuario. El resultado inicial contendrá los actores y casos de uso de alto nivel que describen los requerimientos funcionales en términos generales, para definir los confines y el ámbito del sistema.

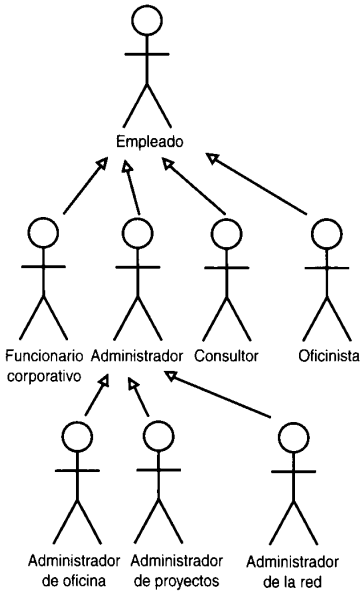
Las entrevistas posteriores profundizarán estos requerimientos y nos dará por resultado modelos de casos de uso que mostrarán los escenarios y las secuencias detalladamente. Esto puede generar nuevos casos de uso que satisfagan las relaciones de inclusión y extensión. Si no logramos comprender bien el dominio, crearemos demasiados casos de uso con demasiados detalles.

Aplicación de los modelos de caso de uso

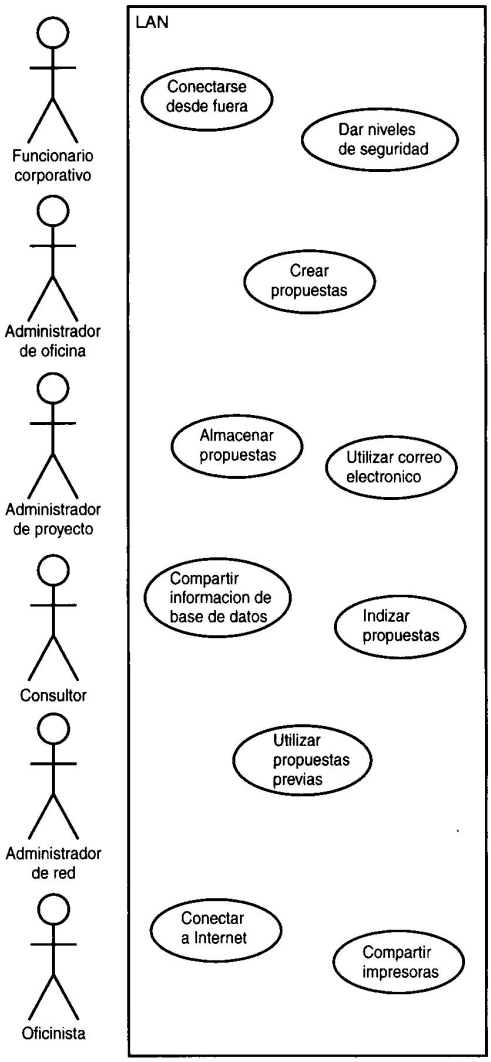
- Comprensión del dominio: Entrevistar al cliente para crear el diagrama de clases.



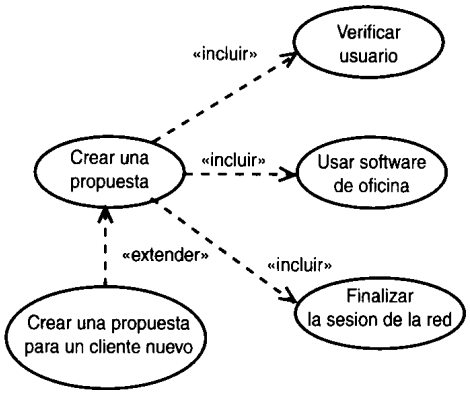
- Comprensión de los usuarios: enfocar a los usuarios para entender los tipos de funcionalidad para crear en el sistema. Podremos mostrar a los usuarios en una jerarquía de generalización.



- Comprensión de los casos de uso: diagramas de aso de uso de alto nivel.



- Profundización: Se generan los modelos para cada caso de uso. En base a las entrevistas, indicaremos cuántos pasos se necesitan en casa caso de uso. Se analiza cada caso de uso y se detalla su modelo.



Dónde estamos

- Elementos estructurales: clases, objetos actores, interfaces, casos de uso son elementos estructurales de UML. Todos representan partes físicas o conceptuales de un modelo.

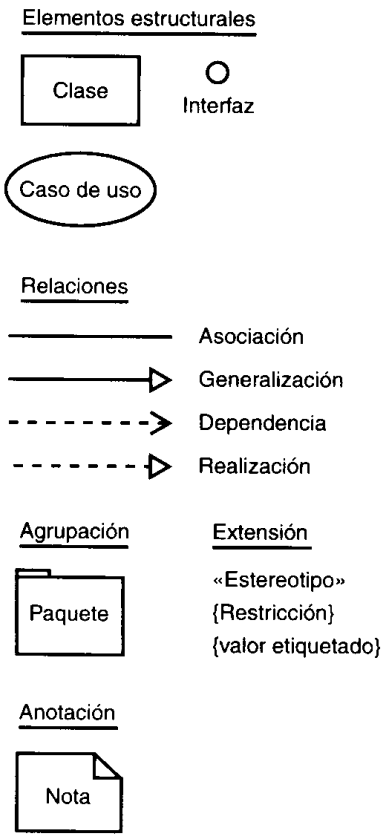
- Relaciones: asociación, generalización, dependencia y realización son las relaciones en UML (inclusión y extensión son tipos de dependencia). Sin relaciones los modelos serían solo listas. Las relaciones conectan los elementos entre sí y conectan los modelos con la realidad.

- Agrupamiento: el paquete es el único elemento de agrupamiento en UML. Nos permite organizar los elementos. Puede contener cualquier tipo de elemento y diferentes tipos también.

- Anotación: La nota es un elemento que nos permite adjuntar restricciones, comentarios, requerimientos y gráficos a los modelos.

- Extensión: los estereotipos nos permiten extender el lenguaje, creando nuevos elementos además de los existentes, para modelar de manera adecuada la sección de la realidad que nos ocupa.

Panorama


Elementos estructurales
Clase
Interfaz
Caso de uso
Relaciones
Asociación
Generalización
Dependencia
Realización
Agrupación
Paquete
Extensión
«Estereotipo»
{Restricción}
{valor etiquetado}
Anotación
Nota

Hora 8 – Diagramas de estados

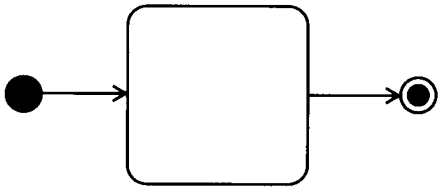
Al contrario de los elementos anteriores, veremos ahora los elementos de comportamiento, que muestran la forma en que las partes de un modelo cambian con el tiempo.

Conforme el sistema interactúa con los usuarios y otros sistemas, los objetos pasan por cambios necesarios para ajustarse a esas interacciones.

Qué es un diagrama de estados

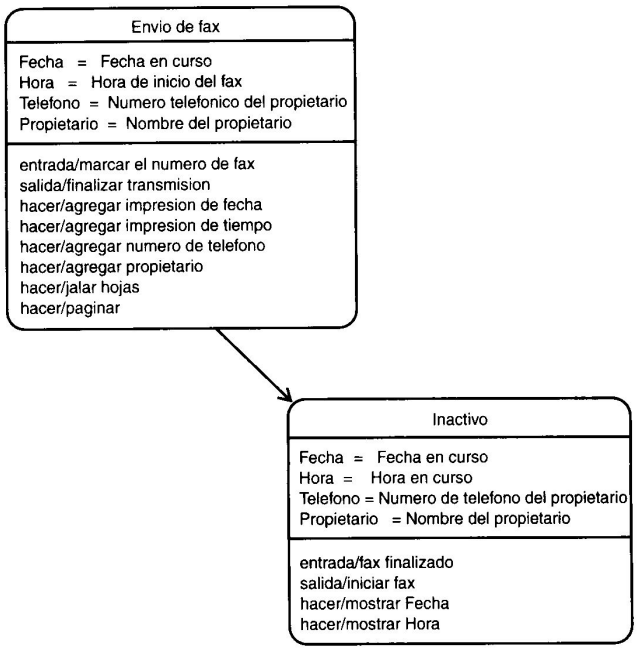
Es un diagrama que captura los cambios de estado de los objetos como consecuencia de ciertos sucesos o del paso del tiempo. Presenta los estados en los que puede estar un objeto y las transiciones entre ellos, así como el punto inicial y final de una secuencia de cambio de estado. Este diagrama presenta las condiciones de un solo objeto, no varios.

Simbología: rectángulo de vértices redondeados que representa un estado, una línea continua y una punta de flecha que representan la transición. La flecha apunta al estado siguiente. El círculo relleno simboliza el punto inicial y el círculo con un punto el final.



Adición de detalles al ícono de estado: podemos dividir el rectángulo en 3. El área superior contiene el nombre del estado, que debe aparecer siempre. El área central, contiene las variables de estado y el área inferior las actividades.

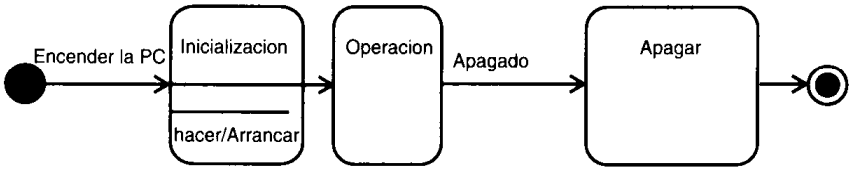
Las variables que sirvan de cronómetro o contadores, son útiles. Las actividades constan de sucesos y acciones. Las más usadas son *entrada*: qué pasa cuando el sistema entra al estado, *salida*: que pasa cuando el sistema sale del estado y *hacer*: qué pasa cuando el sistema está en el estado. Pueden agregarse otros.



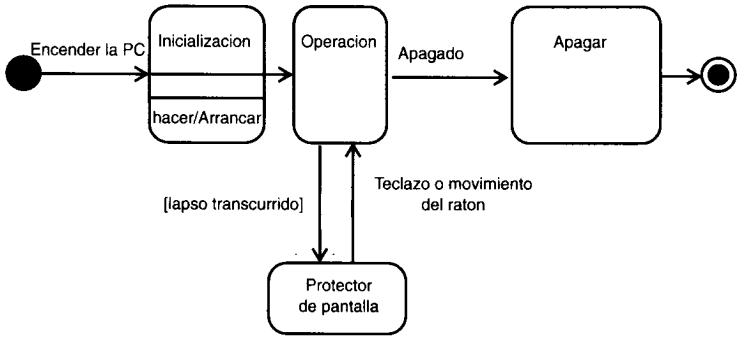
Sucesos y acciones: se agregan a la línea de transición. Podemos indicar un suceso que provoca una transición (la desencadena) y la actividad de cómputo (acción) que se ejecuta y hace que se modifique el estado. Se representan cerca de la línea de transición mediante una diagonal para separar un suceso desencadenado de una acción.

Una *transición no desencadenada* es aquella en la cual un evento la causa sin una acción asociada o cuando una transición se da porque un estado finaliza una actividad.

La interfaz gráfica de usuario (GUI) con la que interactuamos, nos dará ejemplos de estos detalles de transición, en este y los siguientes capítulos. La GUI puede establecerse en 3 estados: **Inicialización, operación y apagar.**



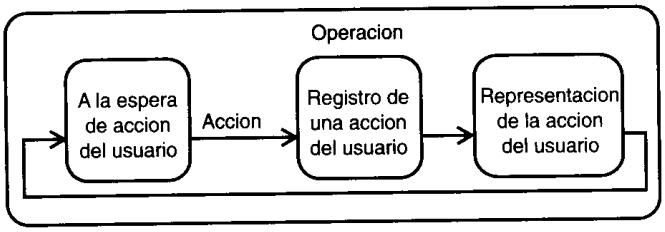
Condiciones de seguridad: se dan cuando ha pasado cierto tiempo sin que haya interacción con el usuario. La GUI hace una transición del estado Operación a otro estado. Se define como una expresión booleana.



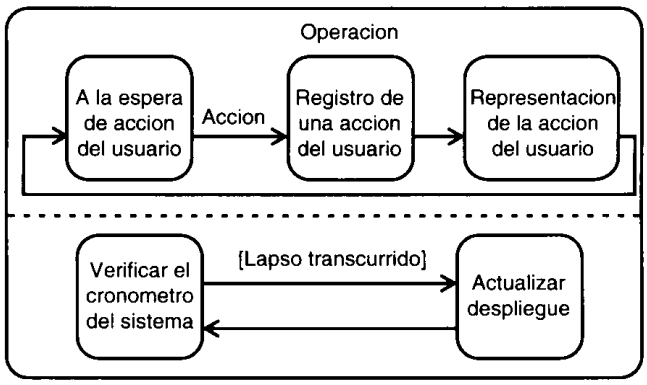
Subestados

Cuando un objeto está en estado Operación, hay muchas cosas que ocurren detrás, aunque no sean evidentes. Por ejemplo, la GUI espera constantemente que el usuario haga algo. Luego registra tales acciones y modifica lo que se presenta en pantalla. Con esto, la GUI atraviesa varios cambios mientras está en operación, estos cambios don de estado y como están dentro de un estado, se conocen como *subestados*. Hay dos tipos:

- Secuenciales: suceden uno detrás de otro. En el caso de la GUI sería: A la espera de la acción del usuario, registro de la acción, representación de la acción. Luego de la secuencia la GUI vuelve a "A la espera de acción del usuario".



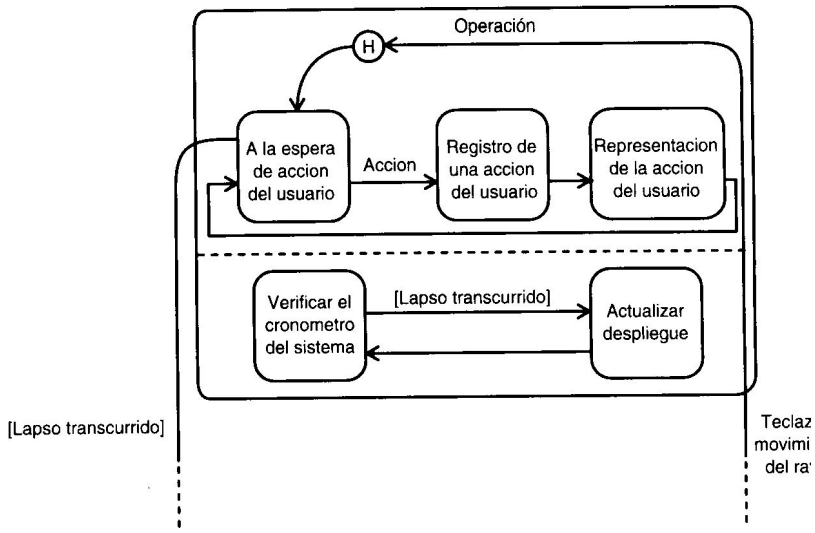
- Concurrentes: puede suceder, que además de la secuencia indicada arriba, la GUI haga otras cosas (por ejemplo, verificar el cronómetro del sistema y actualizar la aplicación luego de un tiempo). Ambas secuencias son concurrentes entre sí. Esto se representa con una línea discontinua entre los estados concurrentes.



El estado *Operación*, es un estado compuesto, ya que consta sólo de subestados secuenciales.

Estados históricos

Un estado compuesto recuerda su subestado activo cuando el objeto trasciende fuera del estado compuesto. El símbolo es la letra "H" encerrada en un círculo conectada con una línea continua al subestado por recordar, con una flecha que apunta a dicho subestado.



Cuando un estado histórico recuerda los subestados en todos los niveles de anidación, se denomina *profundo.* Si sólo recuerda el subestado principal, se llama *superficial*. Uno profundo se representa agregando un * a la "H" en el círculo.

El estado histórico y el inicial, son conocidos como pseudoestados, porque no tienen variables de estado ni actividades.

Mensajes y señales

Los objetos se comunican mediante envío de mensajes. En el ejemplo, el suceso que desencadena es un mensaje de un objeto (usuario) a otro (la GUI). Un objeto que desencadena una transición se denomina *señal*. En orientación a objetos, enviar una señal es igual a crear un objeto señal y transmitirlo al objeto receptor. El objeto señal puede tener atributos y se puede crear una jerarquía de herencia de señales.

Porqué son importantes los diagramas de estados

Permiten a los analistas, diseñadores y desarrolladores comprender el comportamiento de los objetos de un sistema. Al comprender esto, se puede programar mejor el software. Estos diagramas nos permiten saber qué es lo que harán los objetos para evitar sorpresas y producir un sistema que cumpla todo lo requerido.

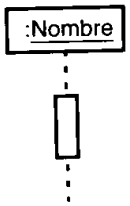
Hora 9 – Diagrama de secuencia

Muestra la forma en que los objetos se comunican entre sí al transcurrir el tiempo.

A diferencia de los diagramas de estados, que se centran en un solo objeto, este diagrama muestra la interacción de varios objetos que se realizan en una secuencia establecida, la cual se toma su tiempo en ir del principio al fin.

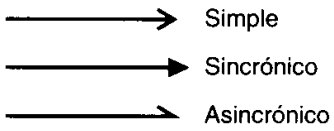
Qué es un diagrama de secuencias

Objetos: se colocan en la parte superior del diagrama de izquierda a derecha. La extensión que hay debajo de cada objeto es una línea discontinua llamada *línea de vida* de un objeto. Junto con la línea se encuentra un pequeño rectángulo conocido como activación que muestra la ejecución de una operación que realiza el objeto. La longitud del rectángulo es la duración de la activación.

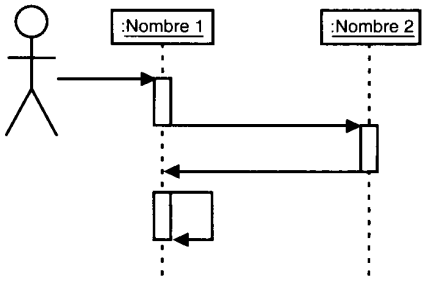


Mensaje: va de un objeto a otro y pasa de la línea de vida de uno a la de otro. Un objeto puede enviarse un mensaje a sí mismo. Puede ser:

- simple: transferencia de control de un objeto a otro.
- sincrónico: espera la respuesta a un mensaje antes de continuar su trabajo.
- asincrónico: no espera una respuesta antes de continuar.



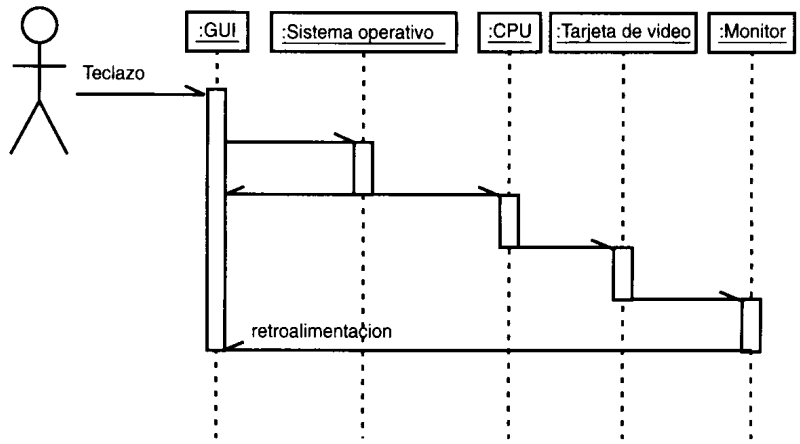
Tiempo: se representa en dirección vertical. Se inicia en la parte superior y avanza hacia la parte inferior. Un mensaje que esté más cerca de la parte superior ocurrirá antes que uno que esté más abajo.



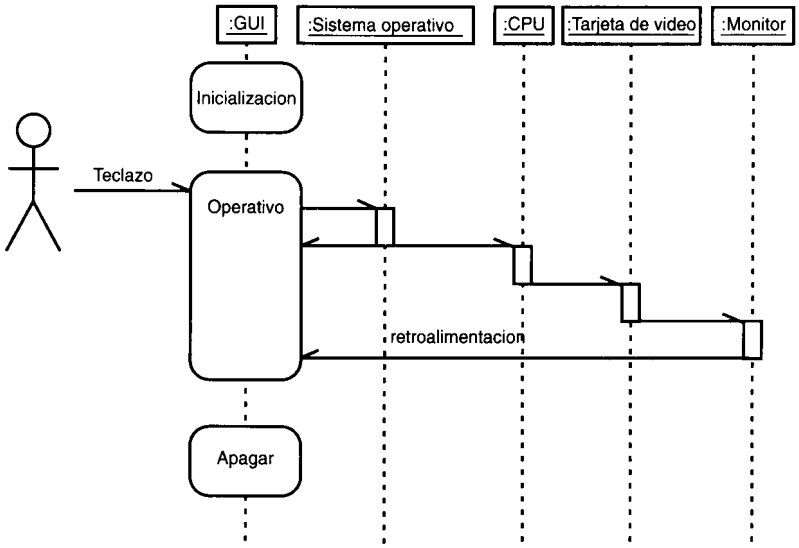
La GUI - La secuencia

1. La GUI notifica al sist. op. que se oprimió una tecla.
2. El sist. op. le notifica a la CPU.
3. El sist. op. actualiza la GUI.
4. La CPU notifica a la placa de video.
5. La tarjeta de video envía un mensaje al monitor.
6. El monitor presenta el carácter alfanumérico en la pantalla.

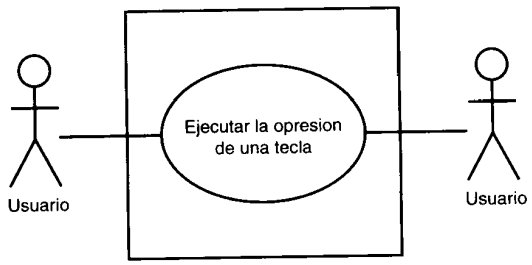
El diagrama de secuencias:



Podemos mostrar los estados de uno o varios objetos en el diagrama. La secuencia se origina y finaliza en el estado operativo de la GUI.



El caso de uso: cuando representamos gráficamente las interacciones de un caso de uso, el diagrama de secuencias delinea el caso de uso dentro del sistema.



Instancias y genéricos

Diagrama de secuencias de instancias: se denomina así al diagrama de secuencia que se centra en un escenario (una instancia) del caso de uso.

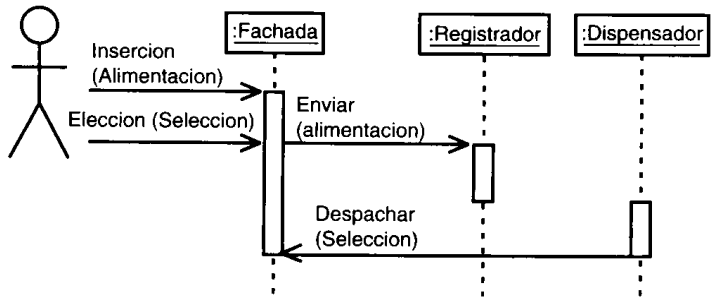
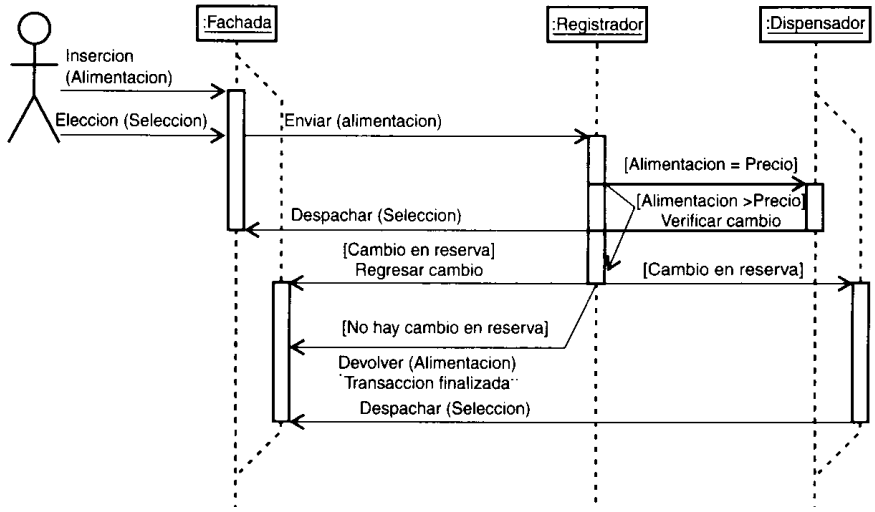


Diagrama de secuencias genérico: cuando tomamos en cuenta todos los escenarios de un caso de uso al momento de crear el diagrama de secuencias. Podemos generar este diagrama a partir del diagrama de secuencias de instancias. Para esto deberemos justificar el control de flujo, es decir, representar las condiciones y consecuencias.

La condición en la secuencia se representa con un "si" (condicional) entre corchetes, arriba de las flechas. Cada condición causa una bifurcación del control en el mensaje, que separa el mensaje en rutinas distintas. Como cada ruta va al mismo objeto, la bifurcación causa una ramificación del control en la línea de vida del objeto receptor y separa las líneas de vida en rutas distintas. En algún lugar de la secuencia, las ramas del mensaje confluyen, como las bifurcaciones en las líneas de vida.

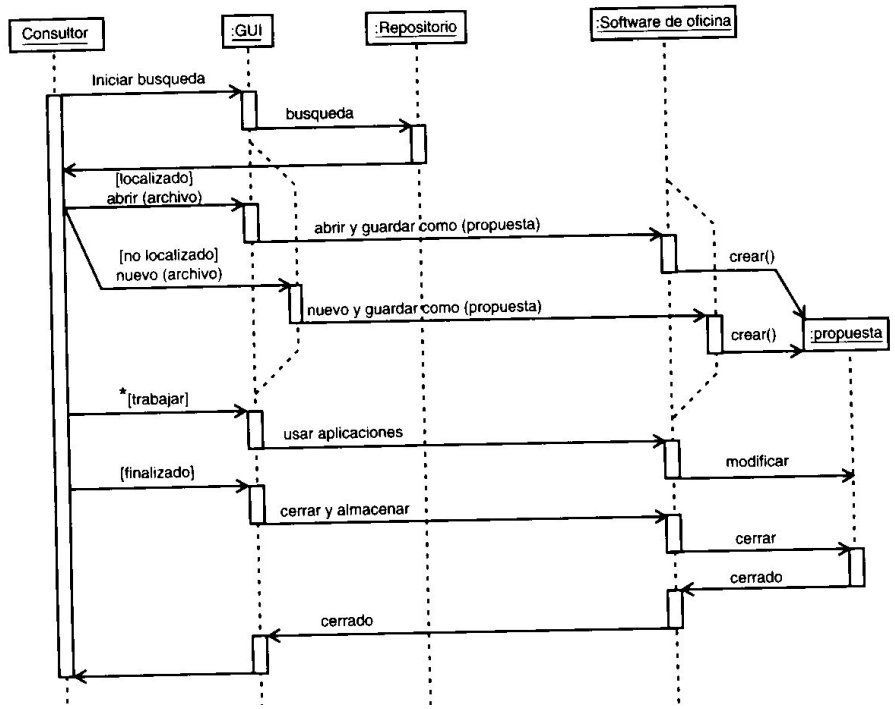


Un diagrama de secuencias está implícito en cada caso de uso.

Creación de un objeto en la secuencia

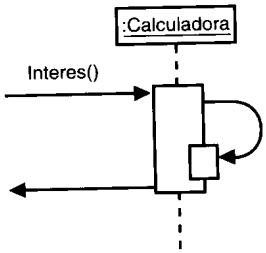
¿Cómo representamos la creación de un objeto en la secuencia de interacciones de objetos? De manera usual, con un rectángulo con nombre. La diferencia es que no lo colocará en la parte superior del diagrama, sino junto con la dimensión vertical, de modo que su ubicación corresponda al momento en que se cree. El mensaje que crea al objeto se llama "Crear()". Los paréntesis indican operación.

En el caso de "mientras", a este control de flujo se lo representa colocando la condición mientras entre corchetes con un asterisco antes del primer corchete.

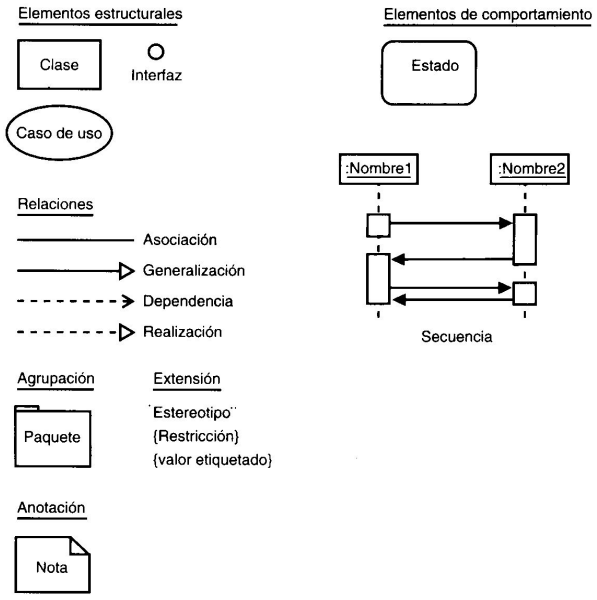


Cómo representar la recursividad

Se dibuja una flecha de mensaje fuera de la activación que significa la operación y un pequeño rectángulo sobrepuesto en la activación. Otra flecha apunta al rectángulo y otra regresa al objeto que inició la recursividad.



Panorama



Hora 10 - Diagrama de colaboraciones

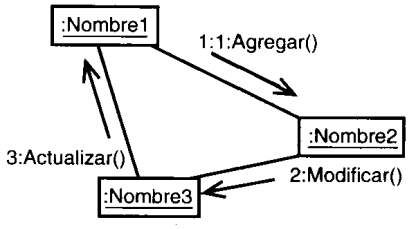
Muestran la forma en que los objetos colaboran entre sí, al igual que el diagrama de secuencia. Ambos representan la misma información y puede convertirse uno en el otro. Un diagrama de colaboración destaca el contexto y organización general de los objetos que interactúan. El diagrama de secuencia se organiza en base al tiempo y el de colaboración en base al espacio.

Qué es un diagrama de colaboración

Es una extensión de uno de objetos. Muestra las relaciones entre objetos y los mensajes que se envían entre sí. Evita la multiplicidad.

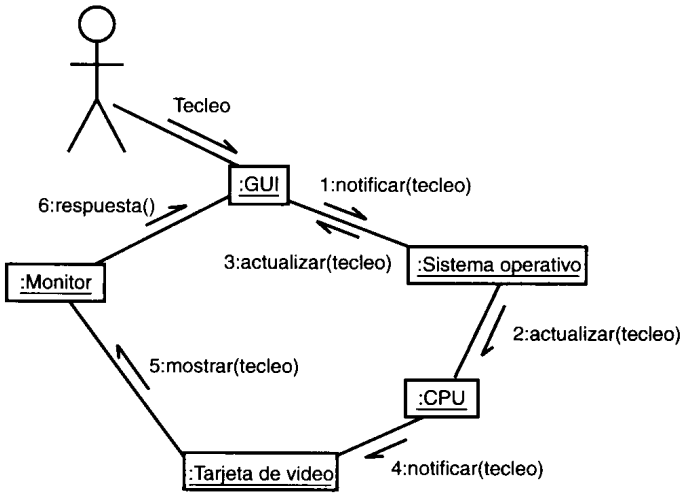
Un mensaje se representa con una flecha cerca de la línea de asociación entre 2 objetos que apunta al objeto receptor. El tipo de mensaje se muestra en una etiqueta cerca de la flecha. El mensaje le indica al objeto receptor que ejecute una de sus operaciones. El mensaje finaliza con un par de paréntesis, donde se colocan los parámetros.

Para convertir un diagrama de secuencias en uno de colaboraciones o viceversa, agregamos una cifra a la etiqueta de un mensaje, que corresponde a la secuencia propia del mensaje. La cifra y el mensaje que separa mediante dos puntos (:).

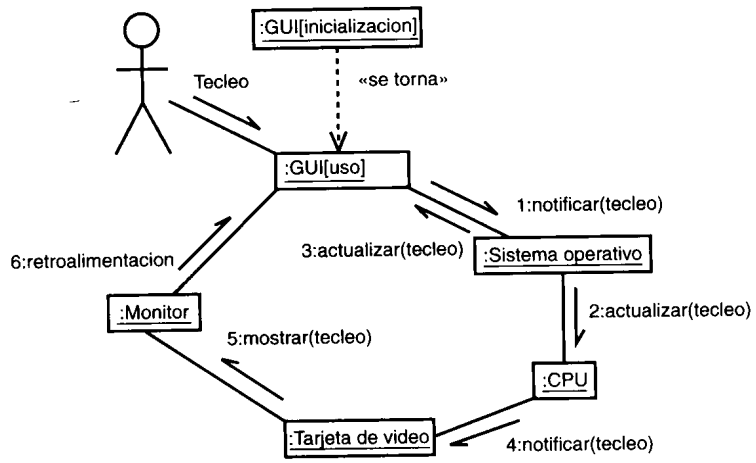


La GUI

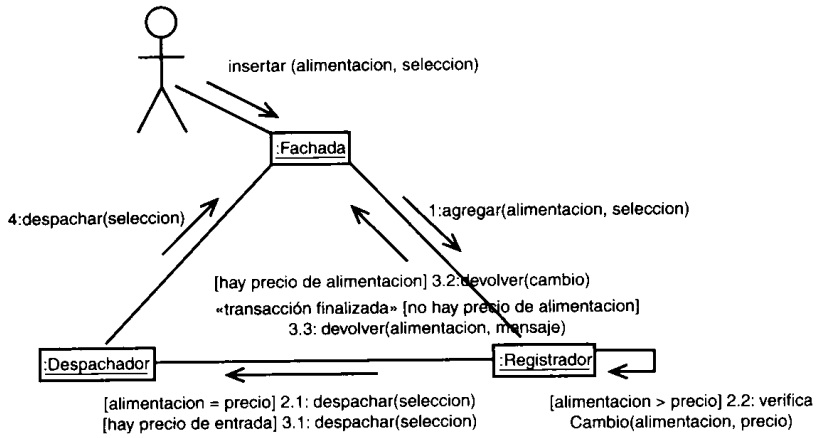
El diagrama inferior muestra la figura agregada que representa al usuario que inicia la secuencia, aunque la figura no es parte de la simbología de este diagrama.



Cambios de estado: el estado se muestra en el rectángulo del objeto. Se agrega otro rectángulo al diagrama que hace las veces de objeto e indica el estado modificado. Los dos se conectan con una línea discontinua y se etiqueta la línea con el estereotipo <<se torna>>.

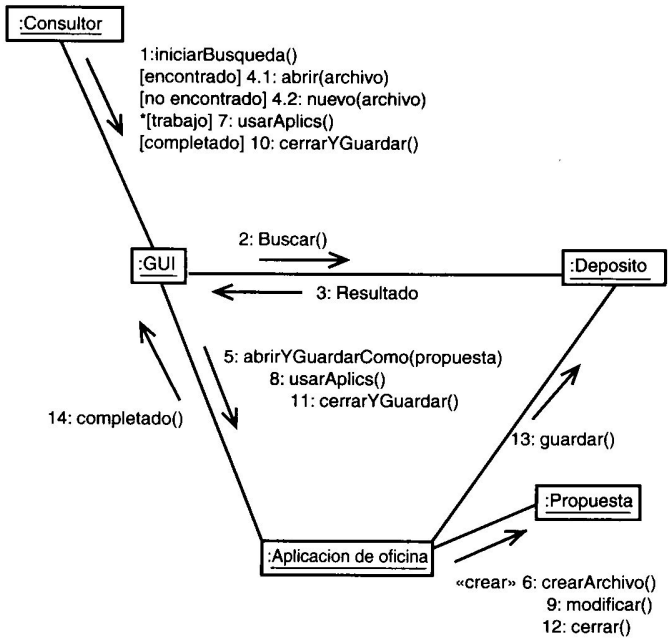


La *anidación* surge cuando tenemos pasos dentro de un paso del diagrama. Cuando agregamos una condición, se agrega una bifurcación en el control de flujo. Numeramos esta bifurcación como un mensaje anidado.



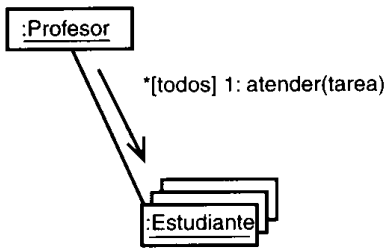
Creación de un objeto

Se muestra agregando un estereotipo <<crear>> al mensaje que genera el objeto. Usaremos instrucciones si (if) y mensajes anidados. También usaremos un ciclo "mientras" (while). Como en el diagrama anterior, lo representamos entre corchetes y antecedemos al del lado izquierdo con un asterisco.

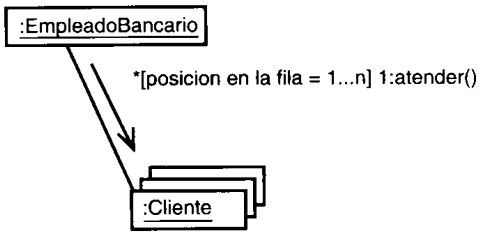


Algunos conceptos más

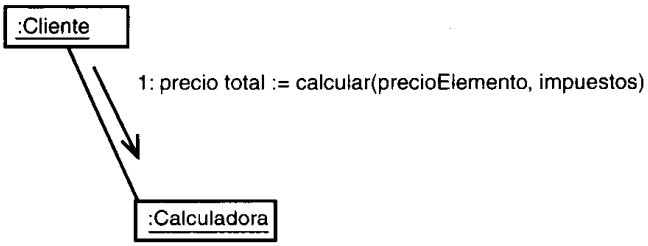
Varios objetos receptores en una clase: un objeto puede enviar mensajes a varios objetos de una misma clase. Estos objetos se representan como una pila de rectángulos que se extienden "desde atrás". Se agrega una condición entre corchetes precedida de un asterisco indicando que el mensaje es para todos los objetos.



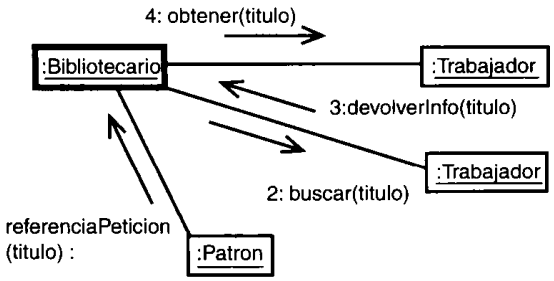
El orden del mensaje puede ser importante. Esto se representa con un "mientras" cuya condición implica orden, junto al mensaje y la pila de rectángulos.



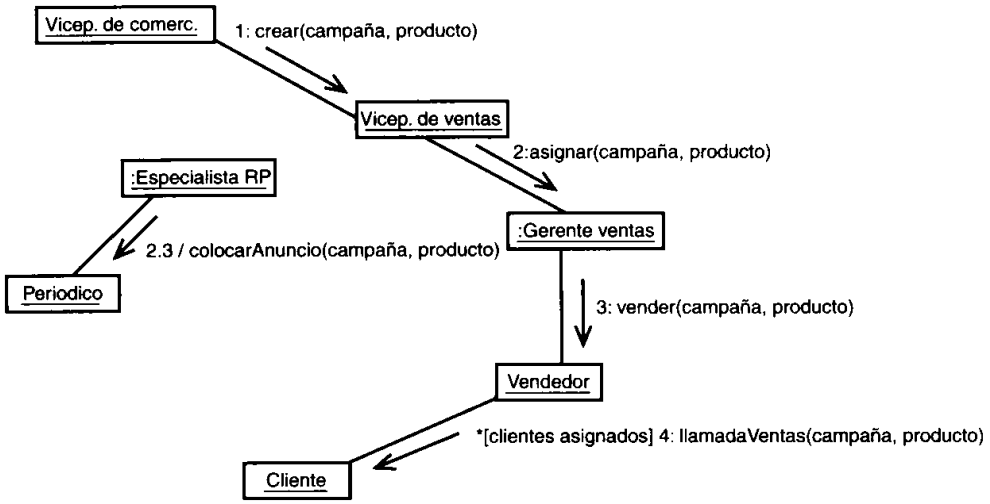
Representación de los resultados: un mensaje podría ser la petición a un objeto para que realice un cálculo y devuelva un valor. Esto se representa escribiendo una expresión que tenga el nombre del valor devuelto a la izquierda seguido de ":=", luego del nombre de la operación y las cantidades con que opera para producir el resultado.



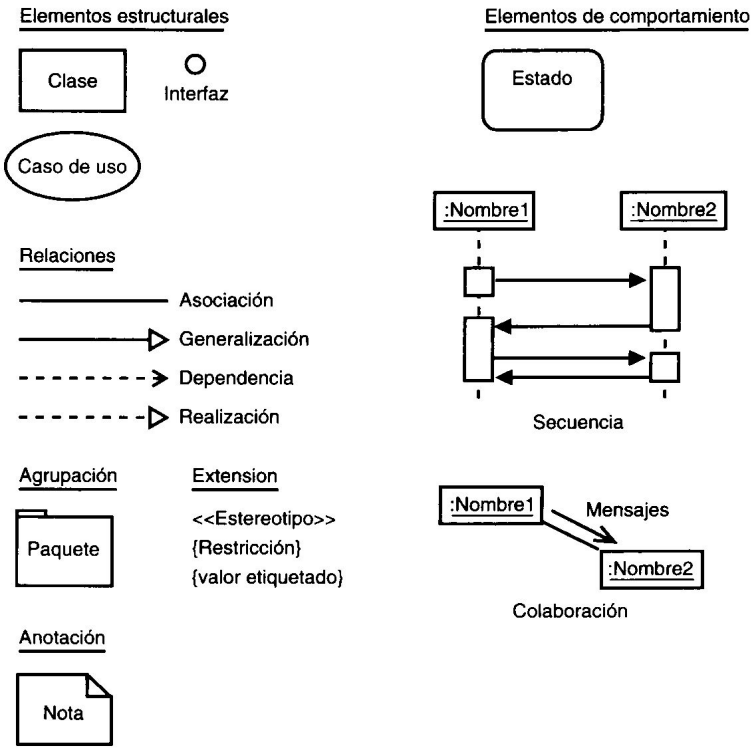
Objetos activos: es un objeto que controla el flujo. Este puede enviar mensajes a los objetos pasivos e interactuar con otros objetos activos. Al proceso de que dos o más procesos activos hagan sus tareas al mismo tiempo se conoce como concurrencia. El objeto activo se representa con el borde grueso y más oscuro.



Sincronización: es cuando un objeto puede enviar un mensaje sólo cuando otros mensajes han sido enviados. El objeto debe sincronizar todos los mensajes en el orden debido. Se representa antecediendo el mensaje con una lista de mensajes que deben completarse antes de que se realice, en vez de la etiqueta numérica. La lista de mensajes se separa con una coma y finaliza con una /.



Panorama


Elementos estructurales
Clase
Interfaz
Caso de uso
Relaciones
Asociación
Generalización
Dependencia
Realización
Agrupación
Paquete
Extension
<<Estereotipo>>
{Restricción}
{valor etiquetado}
Anotación
Nota
Elementos de comportamiento
Estado
:Nombre1
:Nombre2
Secuencia
:Nombre1
Mensajes
:Nombre2
Colaboración

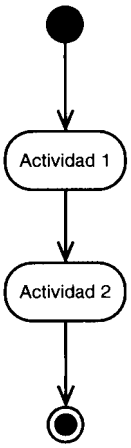
Hora 11 – Diagrama de actividades

Este diagrama muestra los pasos en una operación o proceso. Es similar a un diagrama de flujo. Muestra también puntos de decisión y bifurcación. Es útil para mostrar lo que ocurre en un proceso de negocios u operación.

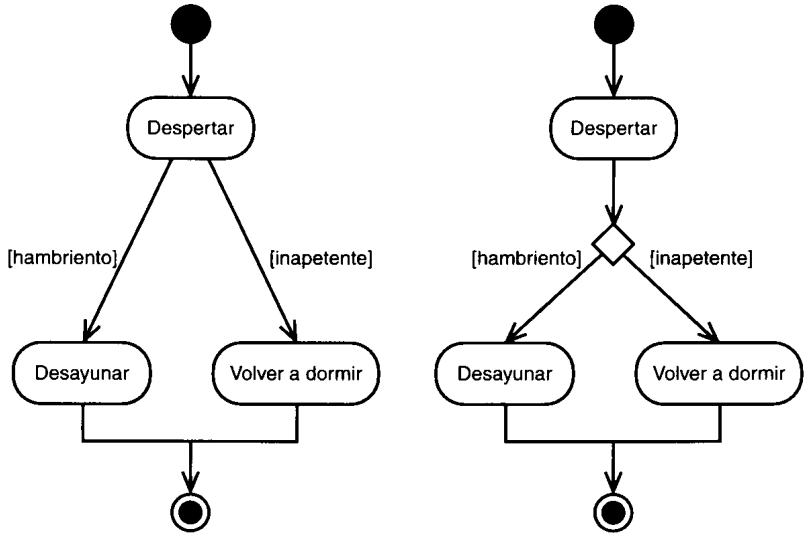
Qué es un diagrama de actividades

Muestra una visión simplificada de lo que ocurre durante una operación o proceso. Es una extensión del diagrama de estados que resalta las actividades intermedias del mismo.

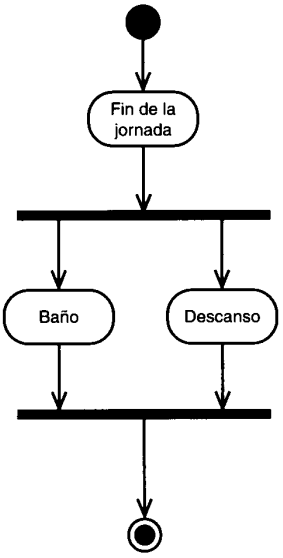
Cada actividad se representa por un rectángulo con las esquinas redondeadas (más angosto y ovalado que el rectángulo de estado). Una flecha representa la transición de una actividad a otra. Se muestra un punto inicial y uno final.



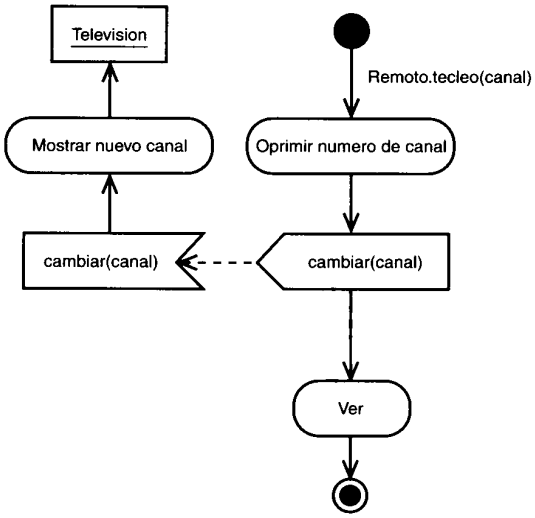
Decisiones, decisiones, decisiones: Casi siempre una secuencia de actividades llega a un punto de decisión. Según la condición se tomará un camino u otro. Esto se representa de dos maneras. O mostrando las rutas posibles que parten directamente de una actividad o llevando la transición a un rombo (elemento del diagrama de flujo) y que de allí salgan las rutas de decisión.



Rutas concurrentes: a veces podremos separar una transición en dos rutas que se ejecutan al mismo tiempo y luego se reúnan. La división se representa con una línea gruesa perpendicular a la transición y las rutas parten de allí. La reincorporación se indica con ambas rutas apuntando a otra línea gruesa.

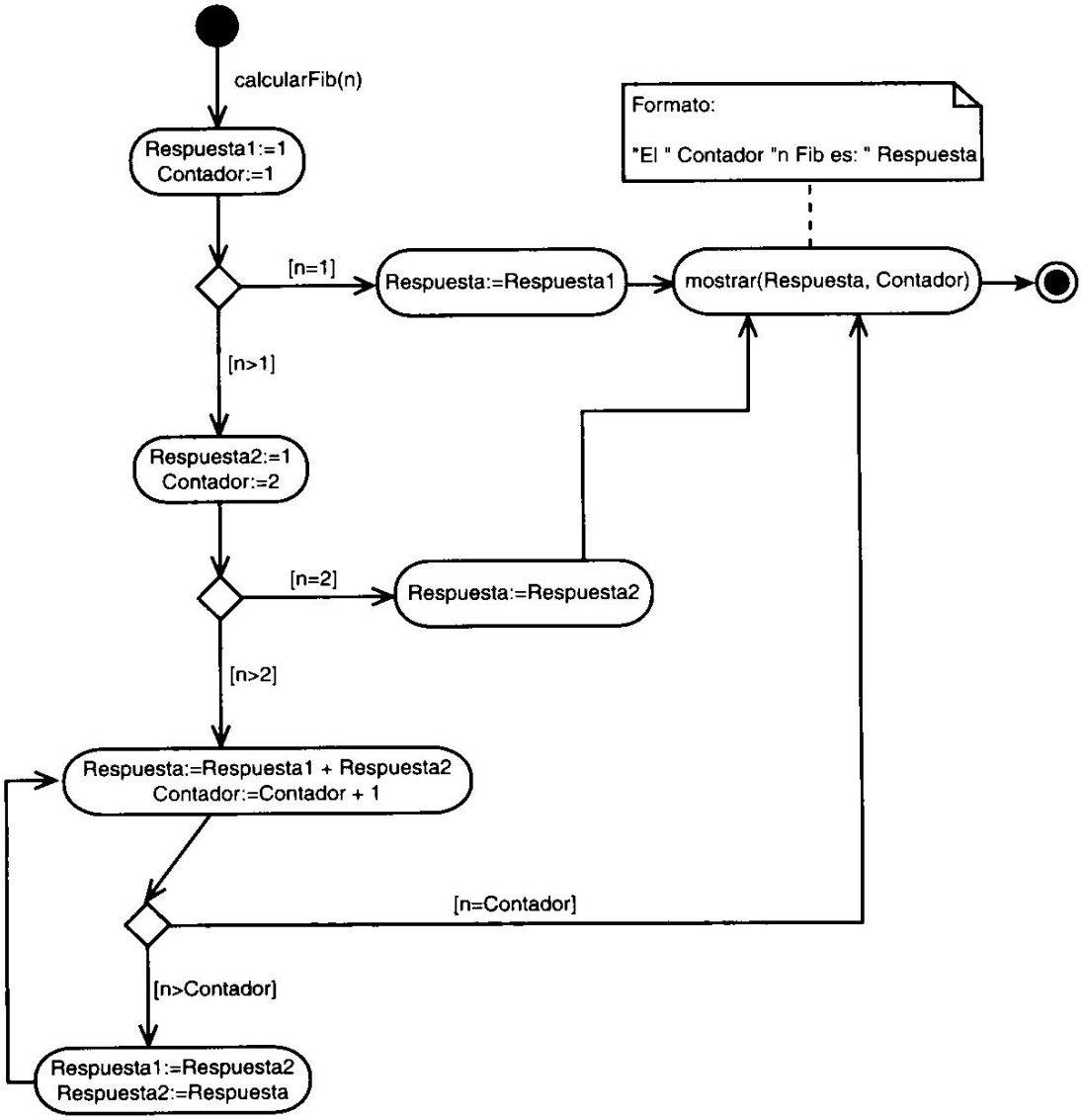


Indicaciones: durante la secuencia de actividades es posible enviar una indicación, que provoca que se ejecute una actividad. Su símbolo es un pentágono convexo y el que la recibe un pentágono cóncavo. En UML el pentágono convexo simboliza el envío de un evento y el cóncavo la recepción del evento.

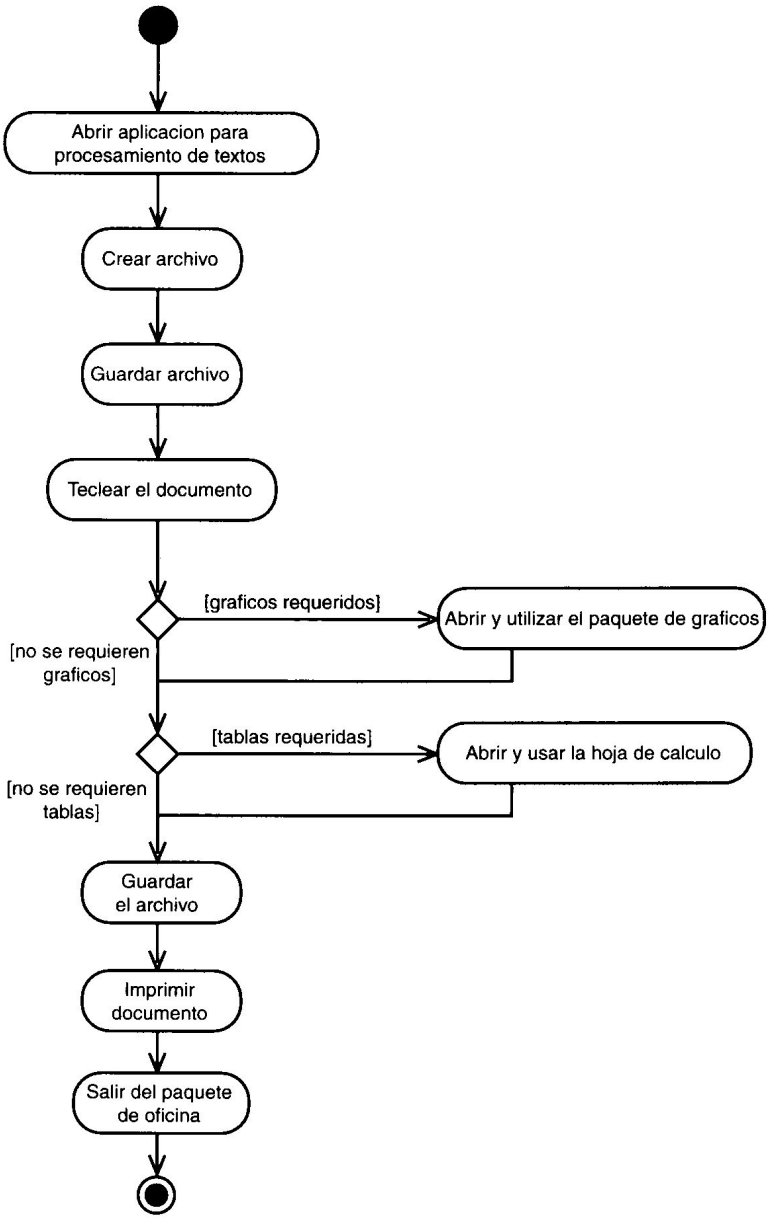


Aplicación de los diagramas de actividades

Operación que calcula la serie de Fibonacci:



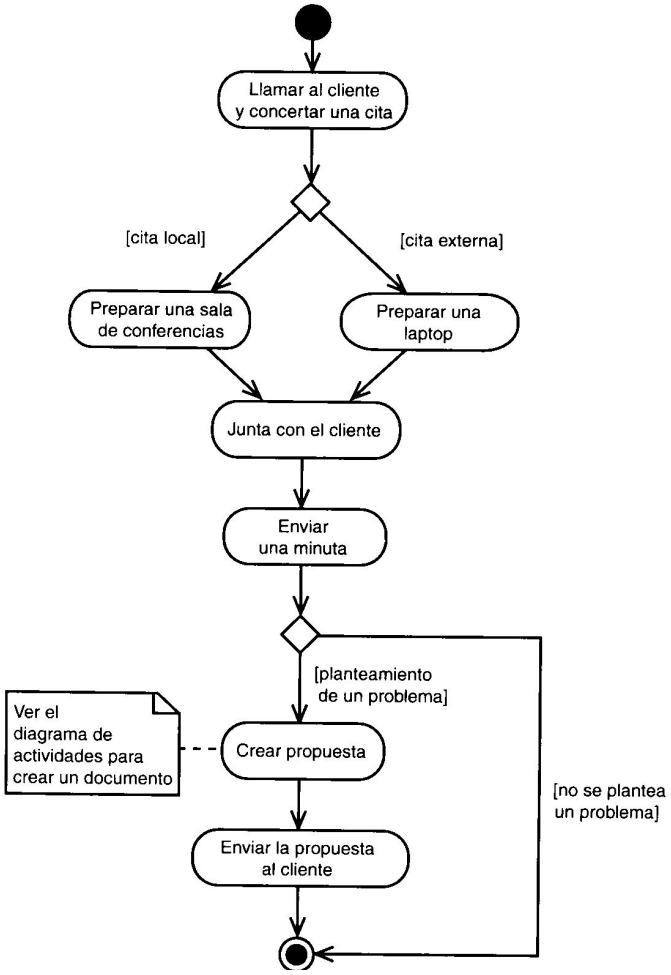
Creación de un documento:

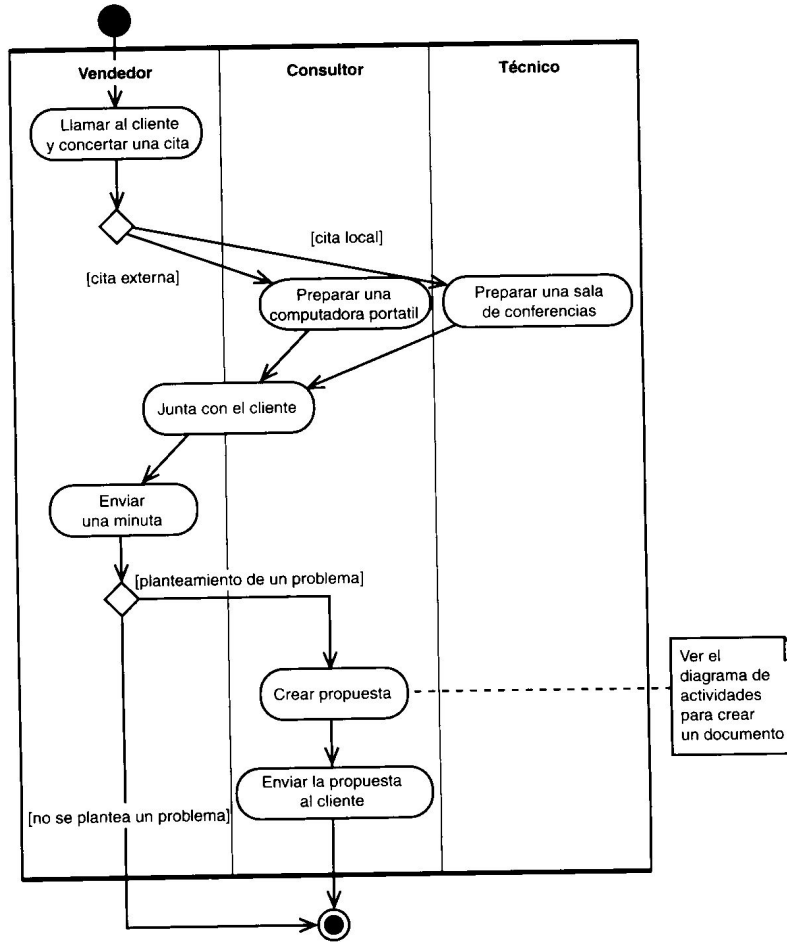


Marcos de responsabilidad

Un aspecto muy útil del diagrama es el poder expandirse y mostrar quién tiene las responsabilidades en un proceso. Para esto debemos separar el diagrama en segmentos paralelos, conocidos como *marcos de responsabilidad*. Cada marco muestra el nombre de un responsable en la parte superior y las actividades de cada uno. Las transiciones pueden llevarse a cabo de un marco a otro.

Abajo veremos un diagrama sin los marcos y el mismo con los marcos.


Llamar al cliente
y concertar una cita
[cita local]
[cita externa]
Preparar una sala
de conferencias
Preparar una
laptop
Junta con el cliente
Enviar
una minuta
[planteamiento
de un problema]
Ver el
diagrama de
actividades para
crear un documento
Crear propuesta
[no se plantea
un problema]
Enviar la propuesta
al cliente


Vendedor
Consultor
Técnico
Llamar al cliente
y concertar una cita
[cita local]
[cita externa]
Preparar una
computadora portatil
Preparar una sala
de conferencias
Junta con el cliente
Enviar
una minuta
[planteamiento de un problema]
Crear propuesta
Ver el
diagrama de
actividades
para crear
un documento
Enviar la propuesta
al cliente
[no se plantea un problema]

Diagramas híbridos

Contienen símbolos que normalmente se asocian con otros diagramas.

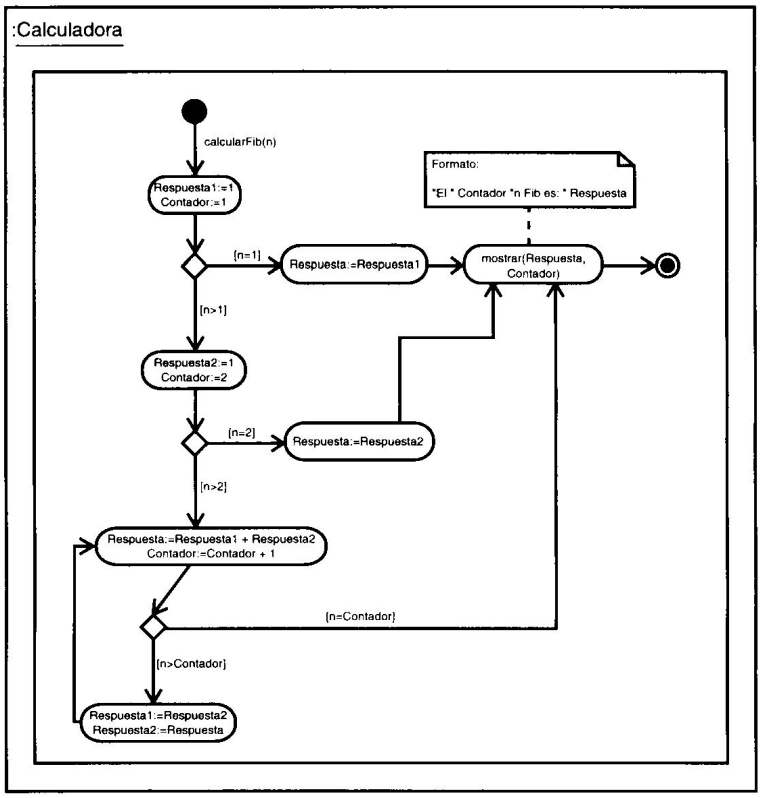
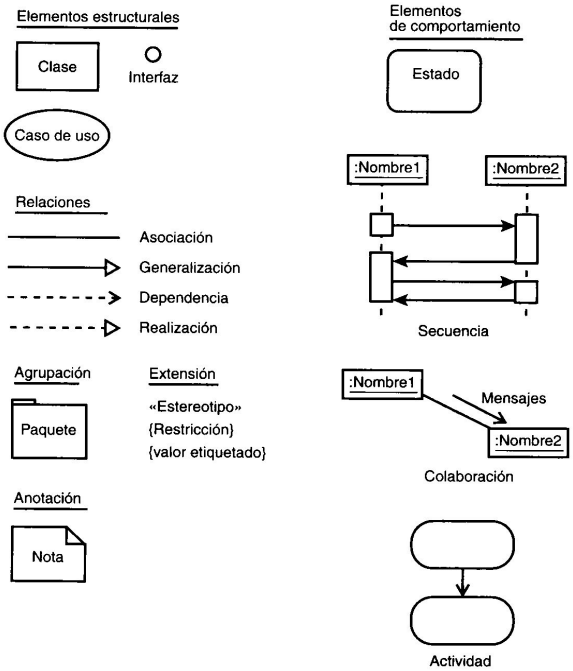


Diagrama de actividades dentro de un objeto.

Panorama



Hora 12 – Diagrama de componentes

Representan entidades reales: componentes de software.

Qué es un componente

Es una parte física del sistema y se encuentra en la computadora. Ejemplo: tabla, archivo de datos, ejecutable, biblioteca de vínculos dinámicos, documentos, etc. Un componente es la personificación en software de una clase, aunque un componente puede implementar más de una clase. Es útil modelar componentes para que:

- Los clientes pueden ver la estructura del sistema final.
- Los desarrolladores tengan una estructura para trabajar en adelante.
- Los que escriben notas técnicas y documenten, sepan de qué escribirán.
- Usted se aliste para reutilizar componentes: si podemos crear componentes para un sistema y reutilizarlo en otro, mejoraremos los tiempos, la productividad y la competitividad empresarial.

Componentes e interfaces

Al tratar con componentes, tratamos con sus interfaces. Refrescamos conocimientos de horas anteriores: algunas clases pueden no estar relacionadas con una clase principal, pero sus acciones pueden incluir operaciones con las mismas firmas. Se pueden reutilizar estas operaciones de clase en clase. La interfaz es, en UML, la que nos permite hacerlo. La interfaz es un conjunto de operaciones que presenta una clase a otra. La relación entre una clase y su interfaz se llama *realización*.

Una interfaz puede ser física o conceptual. La interfaz que usa una clase es la misma que usa su implementación de software (componente). Representaremos igual una interfaz para una clase que para un componente. UML no distingue entre interfaces físicas y conceptuales.

Solo se pueden ejecutar las operaciones de un componente mediante su interfaz. La relación entre un componente y su interfaz también se llama *realización.*

Un componente puede usar los servicios de otro componente. El que sirve, se dice que provee una *interfaz de exportación*, y el que accede, se dice que usa una *interfaz de importación.*

Sustitución y reutilización: podemos sustituir un componente con otro si el nuevo contiene las mismas interfaces que el anterior. También podrá reutilizar un componente en otro sistema si éste puede acceder al componente reutilizado mediante sus interfaces.

Podremos simplificar la vida del desarrollador que intente sustituir o reutilizar un componente si la información de su interfaz está disponible como un modelo.

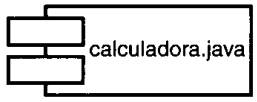
Tipos de componentes

- De distribución: conforman el fundamento de los sistemas ejecutables (dll, ejecutables, controles activex, java beans, etc.).
- Para trabajar en el producto: a partir de los cuales se crean los componentes de distribución (archivos de bd, código, etc.).
- De ejecución: creados como resultado de un sistema en ejecución.

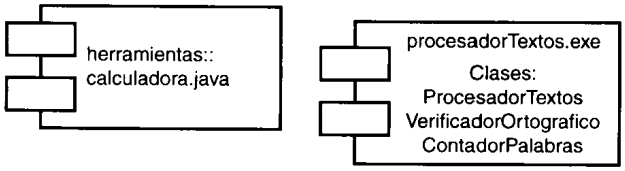
Qué es un diagrama de componentes

Contiene componentes, interfaces y relaciones, así como otros símbolos ya vistos.

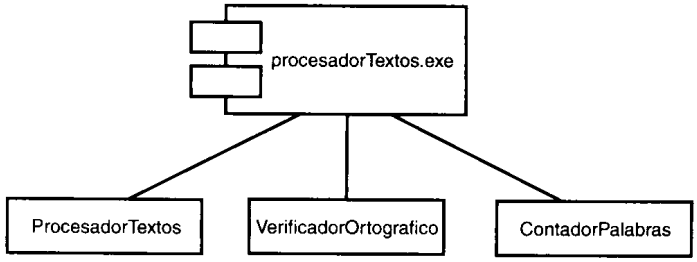
Representación de un componente: rectángulo que tiene otros dos sobrepuestos en su lado izquierdo. El nombre se coloca dentro del símbolo.



Si el componente es miembro de un paquete, podemos usar el nombre del mismo como prefijo para el nombre del componente. También podemos agregar información que detalle al componente.

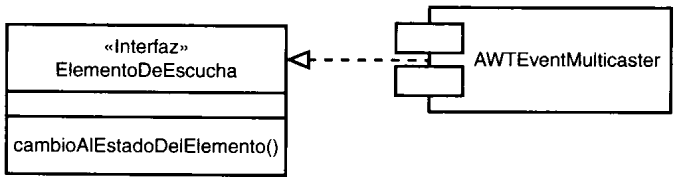


El símbolo de la derecha del diagrama superior, muestra las clases que implementa un componente en particular. Esto se puede representar como en el diagrama de abajo, aunque desordena al diagrama.

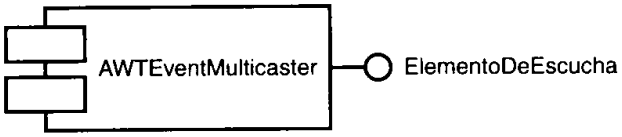


Cómo representar las interfaces:

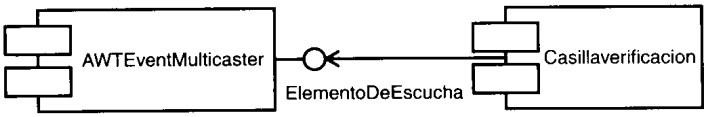
1. Mostrar la interfaz como un rectángulo que contiene la información que se le relaciona y conectarla al componente con una línea discontinua y una punta de flecha con un triángulo sin rellenar que visualiza la realización.

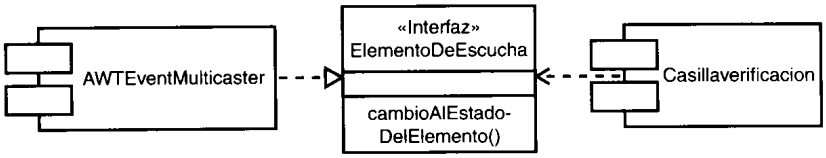


2. Representa la interfaz como un pequeño círculo que se conecta al componente por una línea continua que representa la realización.



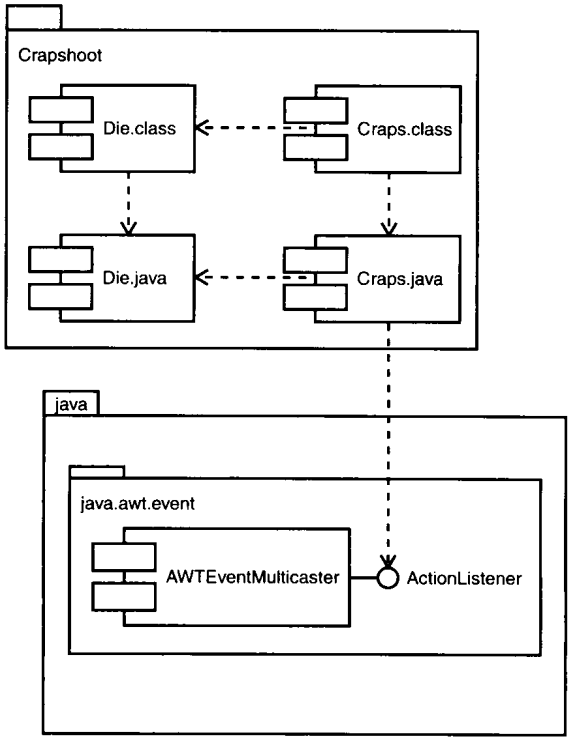
Se puede representar la dependencia, que es la relación entre un componente y una interfaz de importación, con una línea discontinua con una punta de flecha.



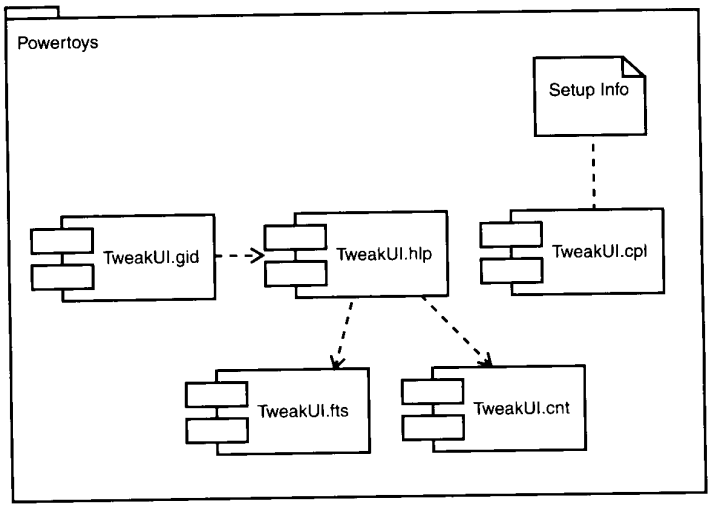


Aplicación de los diagramas de componentes (ejemplos)

Programa en java



PowerToys – TweakUI



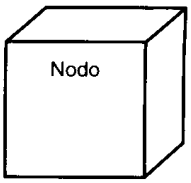
Panorama: el diagrama ya está en el panorama de UML

Hora 13 – Diagramas de distribución

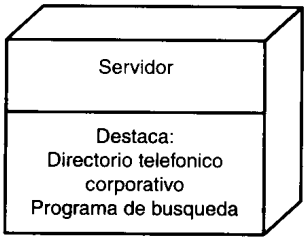
Un diseño sólido de distribución de hardware es básico para el diseño de un sistema. UML nos da los símbolos para crear una imagen clara de la forma en que debe lucir el hardware final.

Qué es un diagrama de distribución

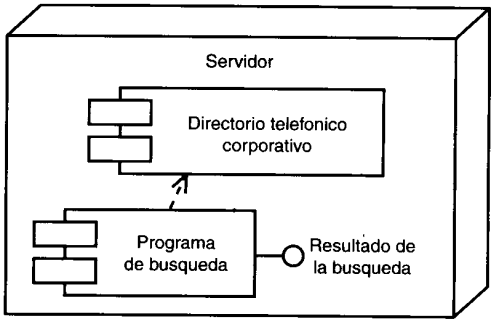
Su elemento primordial es el **nodo** que designa a cualquier tipo de recurso de cómputo. Hay 2 tipos de nodo: el *procesador*, que ejecuta un componente, y el *dispositivo*, que no lo ejecuta. Un dispositivo tiene contacto de alguna manera con el mundo exterior.



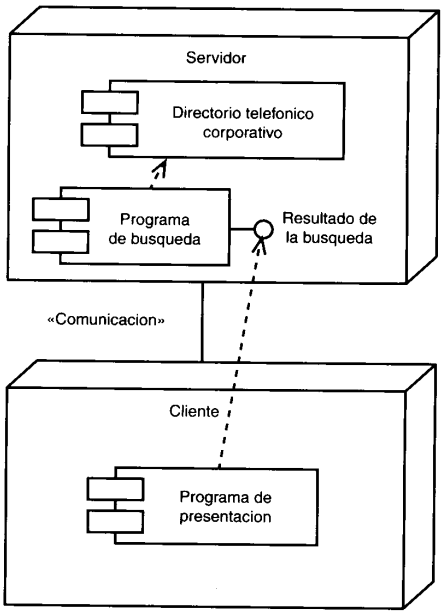
El nombre es una cadena de texto. Si el nodo es parte de un paquete, su nombre puede contener el nombre del paquete. Puede dividir al cubo en compartimentos que agreguen información.



Otra forma de indicar los componentes distribuidos es en relaciones de dependencia con un nodo.



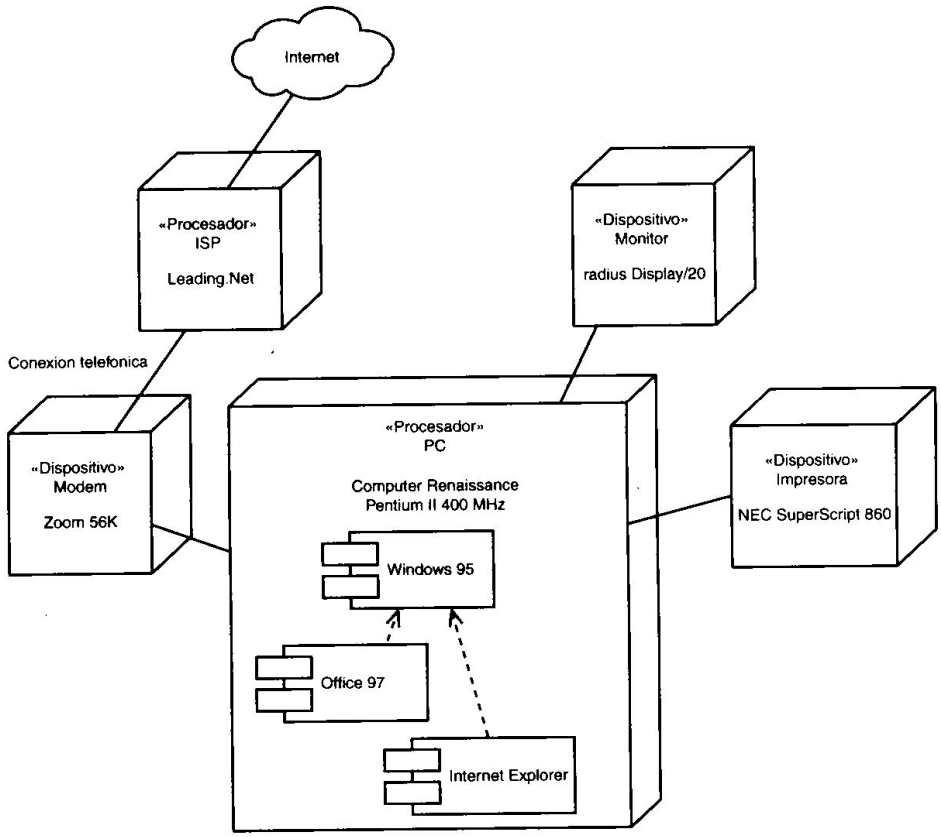
Una línea que asocia dos cubos representa una conexión entre ellos. Podemos usar un estereotipo para dar información respecto a dicha conexión.



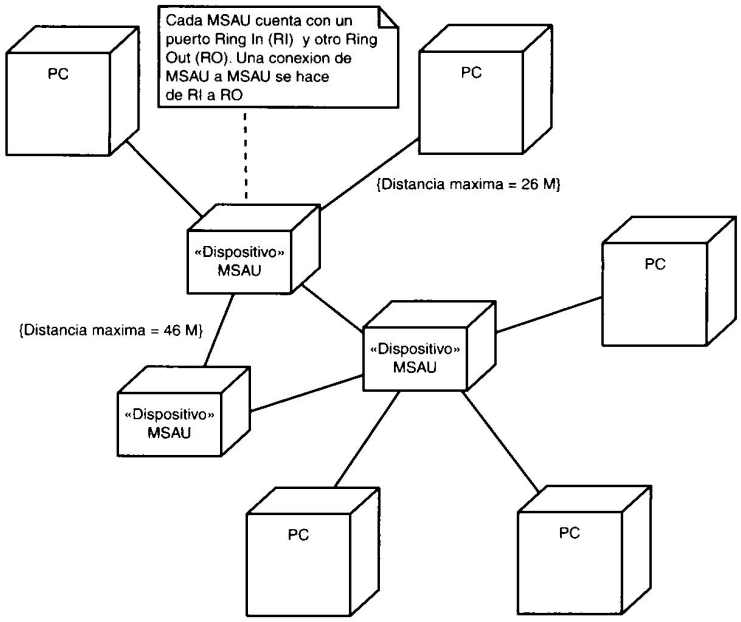
Es posible utilizar agregación, dependencia u otras también. Las representamos de la misma manera que ya se ha visto.

Aplicación de los diagramas de distribución

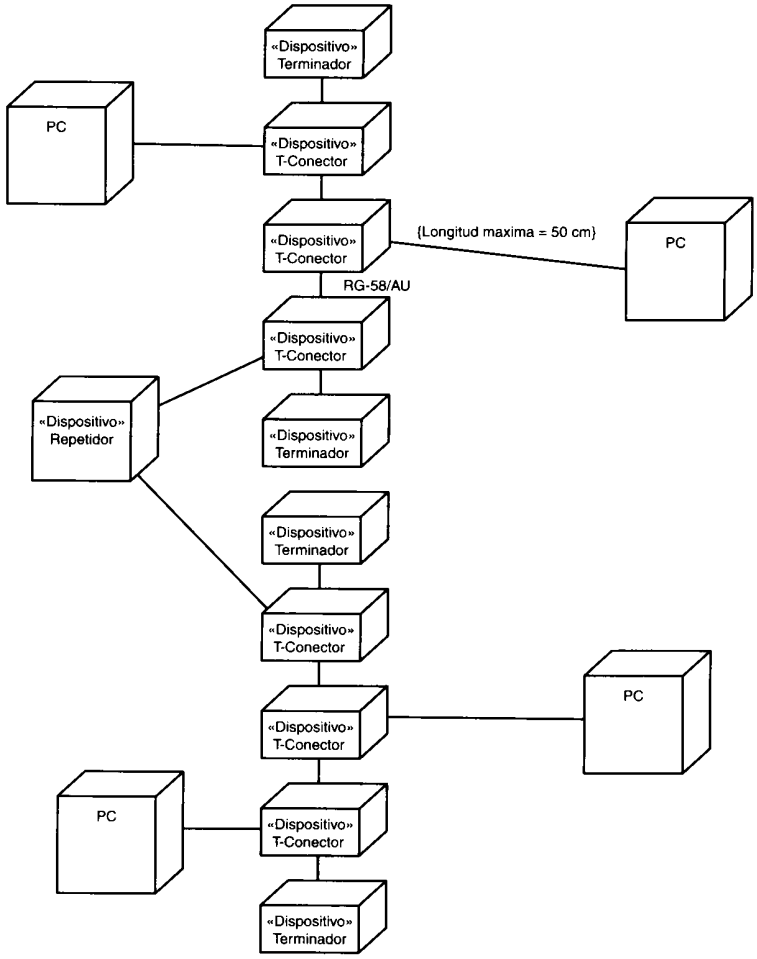
PC doméstica:



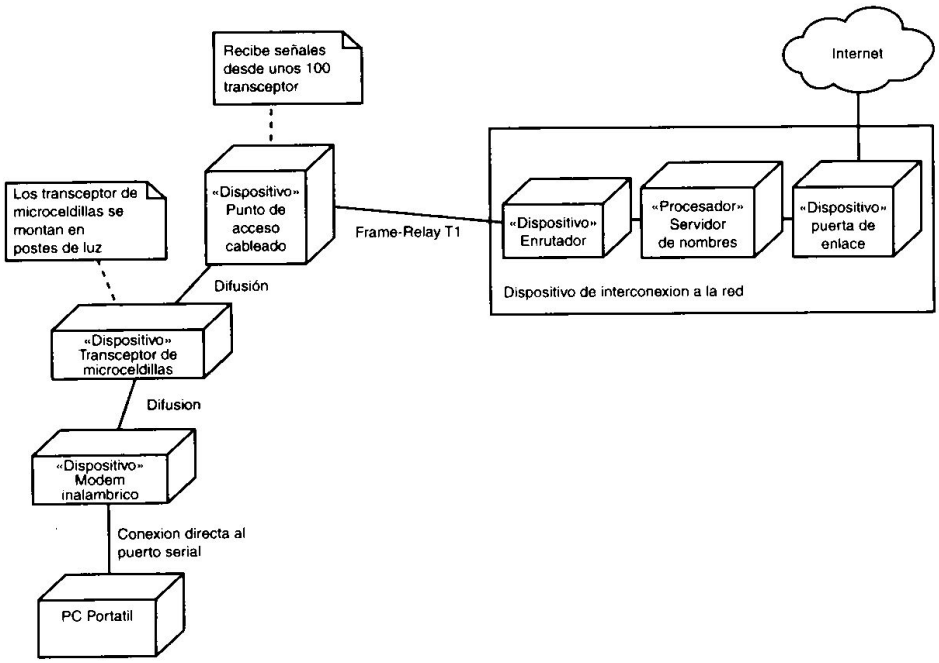
Red Token Ring:



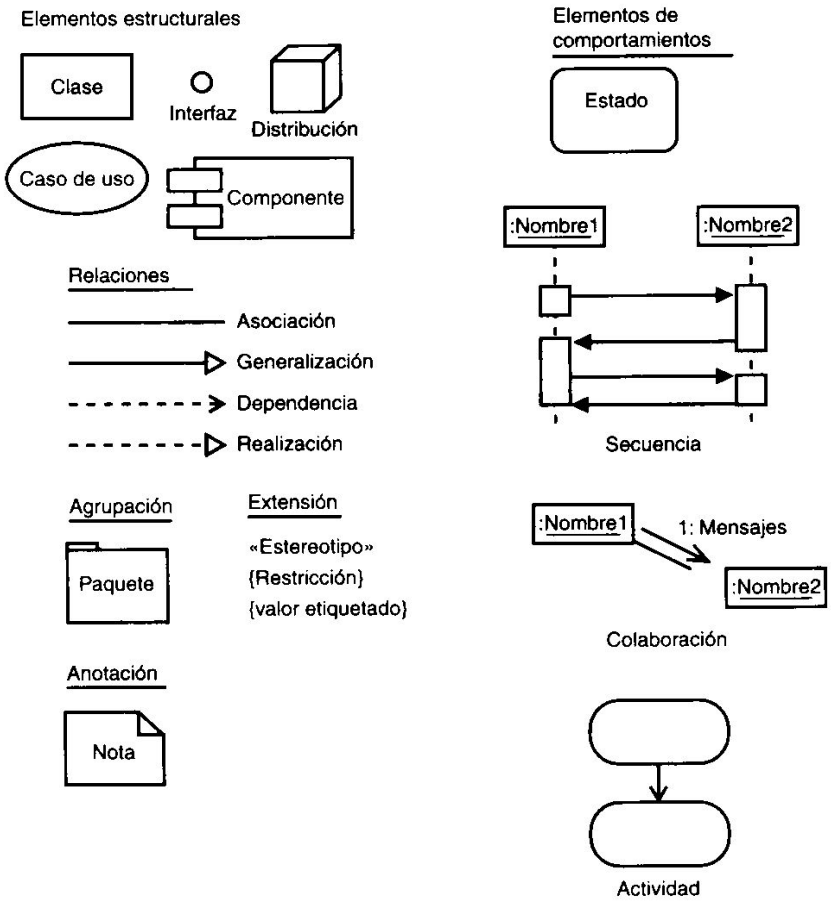
Thin Ethernet:



Red inalámbrica Ricochet de Metricom:



Panorama



Hora 14 – Nociones de los fundamentos de UML

Se ve luego de los otros 13 capítulos, ya que UML es como aprender un idioma extranjero. Lo mejor es sumergirse en el, con lo ya visto y un ejemplo completo, y luego captar las reglas generales, para las cuales ya estaremos preparados.

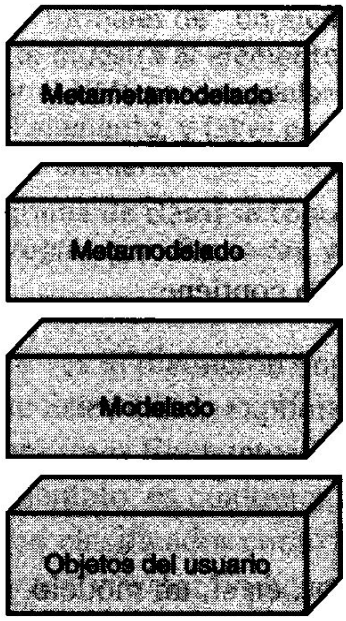
Estructura de UML

Los diagramas nos permiten ver el sistema desde diferentes puntos. Ya que hay varias personas a las que les interesa ver el sistema, debemos ser capaces de comunicar una visión consistente del mismo de diversas formas.

Lo que veremos ahora es la parte formal de UML para asegurar que los elementos ya vistos muestren una idea clara del sistema.

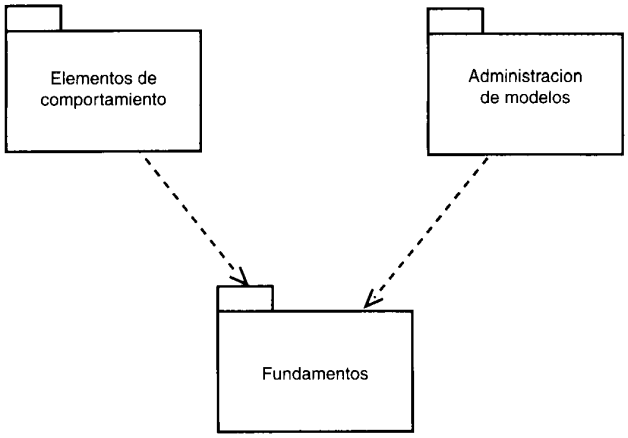
UML tiene una arquitectura de 4 capas. Arquitectura hace referencia a una forma de resumir un conjunto de decisiones para organizar un sistema. Decisiones que se enfocan en los elementos del sistema, es decir, qué son, qué hacen, cómo se comportan, cómo se relacionan y cómo se combinan.

- Capa de objetos del usuario: la vimos cuando seguimos un ejemplo o cuando realizamos un ejercicio. Es la capa más específica, donde el usuario es quien usa UML, no quien usa el sistema.
- Capa de modelado: la vimos cuando vimos las clases. Los primeros estados del análisis tienen que ver con esta capa: trabajar con un experto o un cliente para obtener información del dominio y con un usuario para comprender los casos de uso que se tienen que tomar en cuenta.
- Capa de metamodelado: la vimos cuando aprendimos conceptos nuevos. Esta capa define el lenguaje para un modelo específico.
- Capa de metametamodelado: es una forma de definir un lenguaje que especifique todos los elementos de UML. Pocas veces el analista trata con esta capa. Se denomina sí porque define lo que va en el metamodelado.

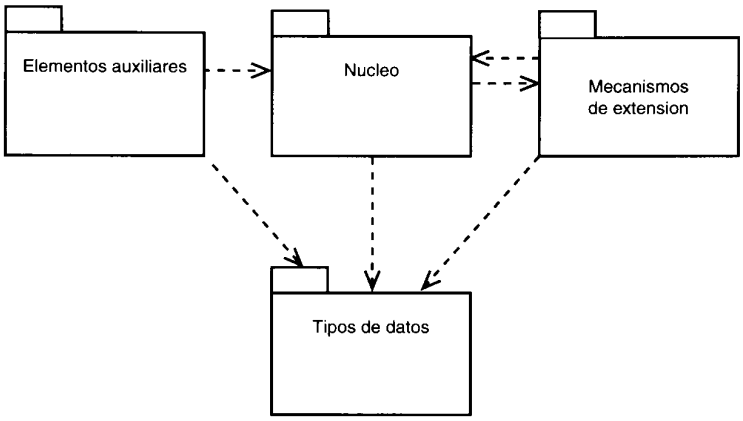


Capa del metamodelado: cercano y personal

Esta capa es la base del panorama que ya vimos en los otros capítulos. Está formada por 3 paquetes:



- Fundamentos: Contiene 4 paquetes: Núcleo, elementos auxiliares, tipos de datos y mecanismos de extensión.



Núcleo: define lo que necesita para crear un modelo UML. Cada uno de los elementos definidos es *abstracto*, es decir, no se crean instancias de ellos o *concreto*, es decir, sí se crean instancias.
Abstractos: ElementoDeModelo, ElementoGeneralizable y Clasificador.
Concretos: Clase, Interfaz, Asociación y TipoDeDatos.

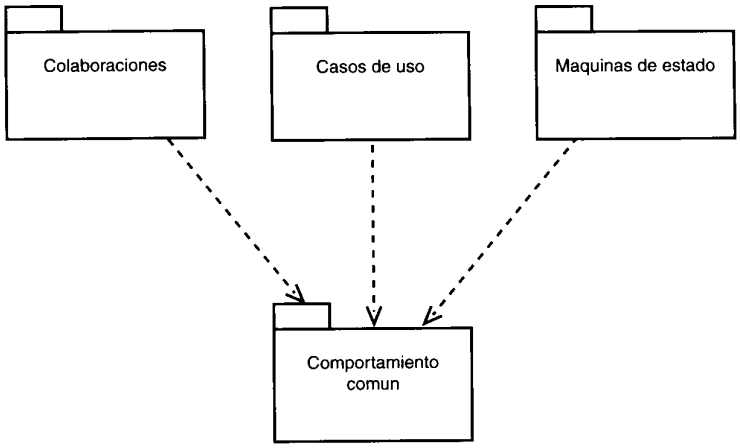
Los elementos concretos se derivan de los abstractos, por lo tanto hablamos de clases y herencia. En este caso las denominamos *metaclases*. Por lo tanto un clasificador es una metaclase.

Elementos auxiliares: define la Dependencia, Componentes y Nodos, entre otros.

Tipo de Datos: especifica los tipos de datos que UML usa, incluyendo los tipos primitivos y enumeraciones (listan los valores posibles).

Mecanismos de Extensión: especifica cómo puede extender a UML e incluye algunas extensiones ya hechas.

- Elementos de comportamiento: consta de 4 paquetes: colaboraciones, casos de uso, máquinas de estado y comportamiento común.



  Comportamiento común: proporciona los conceptos de los elementos dinámicos y soporta a los otros paquetes: casos de uso, máquinas de estado y colaboraciones. Estos "conceptos" incluyen: Seña, Enlace y Punto final de asociación.

  Colaboraciones: abarca más que los diagramas de colaboración. Aquí, una colaboración describe la forma en que los clasificadores y sus asociaciones trabajan en conjunto para realizar una tarea. Los clasificadores conforman al *contexto* de la colaboración: las asociaciones conforman la *interacción*.

  Casos de uso: detalla los conceptos de los casos de uso (actor, caso de uso, etc.). Ambos conceptos son clasificadores. El objetivo general es poder describir el comportamiento de un sistema sin entrar en detalles.

  Máquinas de estado: da los detalles formales de los conceptos de los diagramas de estados y de actividad.

- Administración de modelos: define al *Modelo, Subsistema* y *Paquete*. La meta de estos elementos es agrupar los *ElementosDeModelo* de todo tipo.

Extensión de UML

Podemos pulir los diagramas con detalles que expliquen mejor su significado (estereotipos, restricciones y valores etiquetados). Podemos crear una extensión sobre la marcha para añadir al modelo cuestiones e ideas importantes del dominio.

Estereotipos

Sirven para extender un elemento de UML para que sea una instancia de una nueva metaclase y se escriben entre dos pares de mayor y menor (<< >>). Esto agrega flexibilidad, ya que podremos usar un elemento existente como base para crear nuevos elementos, que capturen algún aspecto propio del dominio de nuestro sistema de una manera que UML no podría hacerlo.

Las herramientas de modelado, tienen que almacenar y manejar las clases para generar código e informes. El mecanismo del estereotipo les permite hacerlo con nuestras propias creaciones.

Tipos de estereotipos generados:

- Dependencia: puede tomar la mayor cantidad de estereotipos ya creados. Cada uno extiende una relación de dependencia entre un origen y un destino.

  <<se convierte en>>: muestra que el origen y el destino son el mismo objeto en distintos momentos.
  <<llamar>>: tiene una operación como origen y una como destino.
  <<copiar>>: el destino es una copia exacta del origen.
  <<derivar>>: el origen se deriva en el destino.
  <<reunir>> o <<friend>>: el origen tiene acceso al destino sin importar la visibilidad.
  <<extender>>: los comportamientos del caso de uso de destino se agregan al de origen.
  <<usar>>: algunos casos de uso tienen cierto comportamiento en común. Esto permite usar dicho comportamiento sin repetirlo una y otra vez.
  <<importar>>: agrega el contenido del destino al espacio de nombres del origen.
  <<instancia>>: el origen es una instancia del destino, que siempre es un clasificador. Es una dependencia de <<metadestino>>: tanto el destino como el origen son clasificadores y el destino la metaclase de origen.
  <<enviar>>: el origen es una operación y el destino una señal.

- Clasificador: Los estereotipos extienden a los clasificadores de varias maneras.

  <<metaclase>>: el clasificador al que está adjunto es una metaclase de otra clase.
  <<tipodeautoridad>>: un clasificador tiene objetos que provienen de un antecesor en particular. También se usa en una dependencia para mostrar que el destino es un tipo de potestad del origen.
  <<proceso>> y <<subproceso>>: indican que su clasificador es una clase activa, es decir, sus objetos pueden iniciar la actividad de control.
  <utilería>>: es una colección titulada de atributos y operaciones que no son miembros de tal clasificador: un clasificador que no tiene instancias.
  <<estereotipo>>: el clasificador funciona como un estereotipo y permite modelar jerarquías de estereotipos.

- Clase: son más específicos que en los clasificadores.

  <<tipo>>: es una clase que establece un dominio de objetos junto con sus atributos, operaciones y asociaciones. No contiene métodos (algoritmos ejecutables).
  <<claseDeImplementacion>>: es lo contrario a <<tipo>>. Representa la implementación de una clase en un lenguaje de programación.

- Generalización: relación entre clasificadores con su propio conjunto de estereotipos.

<<heredar>>: las instancias del subtipo no pueden sustituirse por instancias del supertipo.
<<subclase>>: las instancias de la subclase no son sustituible por las instancias de la superclase.
<<privado>>: herencia exclusiva, oculta los atributos heredados y operaciones de una clase a sus ancestros.

- Paquete: son estereotipos directos.

  <<fachada>>: es un paquete que contiene referencias a elementos de otros paquetes y no tiene elementos propios.
  <<sistema>>: es una colección de modelos de un sistema.
  <<cabo>>: es un paquete que proporciona solo las partes públicas de otro paquete.

  Un paquete puede incluir patrones. Un patrón es un tipo de colaboración entre los elementos que ha probado su efectividad en diversas situaciones. <<marcoDeTrabajo>> es un paquete estereotipado que solo contiene patrones.

  Ya que los paquetes pueden tener otros paquetes adentro, nos sirve tener un estereotipo que indique cuál paquete está en el nivel superior. Este es el <<paqueteDeNivelSuperior>>.

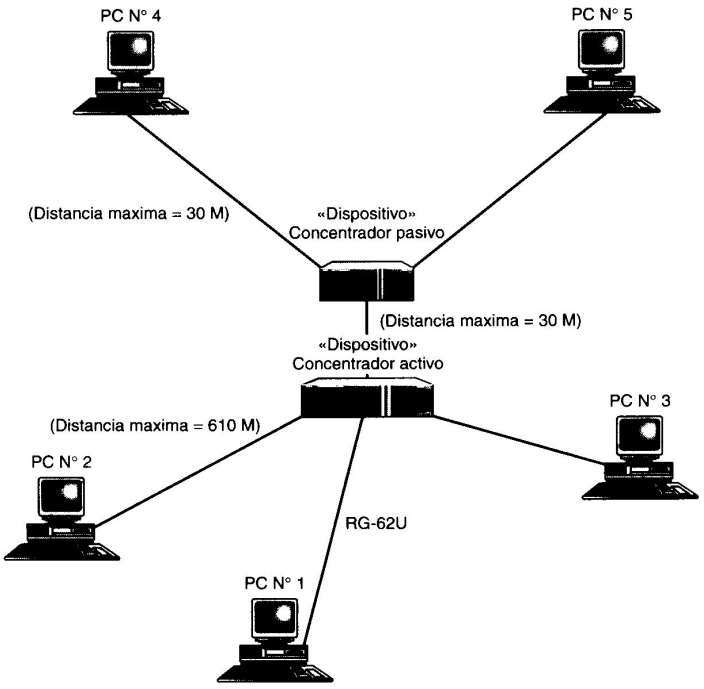
- Componente: son muy directos. Podemos mostrar que un componente es un documento, un ejecutable, un archivo, una tabla de datos o una biblioteca. Los estereotipos son <<documento>>, <<ejecutable>>, <<archivo>>, <<tabla>> y <<biblioteca>>.

- Algunos otros estereotipos: un comentario que aparece en una nota adjunta puede tener un estereotipo <<requerimiento>> que indica que el comentario establece un requisito para el elemento adjunto.

  Dentro de una clase una operación puede crear o destruir una instancia. Tales características se indican mediante <<crear>> y <<destruir>>.

  Las restricciones también funcionan con estereotipos. <<condicionPrevia>> o <<condicionPosterior>> se usan para indicar justamente las condiciones de una operación. A veces podremos adjuntar una restricción a un conjunto de clasificadores o relaciones y necesitaremos indicar que las condiciones de la restricción deberán tener todos los clasificadores, relaciones e instancias. Para esto usamos <<invariable>>.

- Estereotipos gráficos: por ejemplo, figuras de procesadores o dispositivos que reemplacen a los cubos de un diagrama de distribución.



Restricciones

Se encuentran entre llaves. Indican condiciones para las asociaciones, extremos de vínculos, generalizaciones y peticiones (transmisión de señales o llamadas a operaciones).

La restricción {o} se aplica a un conjunto de asociaciones y muestra que solo una puede usarse. Otra restricción basada en asociaciones, {implícito}, indica que una asociación es conceptual.

Los extremos de vínculos que son puntos finales de vínculos entre objetos, pueden contener muchas restricciones. Cada una indica el porqué el objeto en el extremo del vínculo es visible:

{parámetro}: muestra que el objeto es un valor necesario relativo al vínculo.
{propio}: indica que el objeto es el despachador de una petición.
{global} y {local}: indican el ámbito de un objeto respecto al vínculo.
{asociación}: denota que el objeto es visible por su coalición.

Un conjunto de generalizaciones puede ser:
{completo}: todos su subtipos han sido especificados.
{incompleto}: aún pueden agregarse subtipos.
{traslapado}: más de un subtipo puede funcionar como un tipo de instancia.
{desarticulado}: sólo un subtipo puede ser un tipo de instancia, lo que es predeterminado en la generalización.

Si una petición se envía a diversas instancias de destino, en un orden no especificado, es una {difusión}. Si varias instancias devuelven valores y la mayoría de los mismos determinan un solo valor, la restricción será un {voto}.

Valores etiquetados

Se escriben entre llaves. Es una *etiqueta* un = y un valor.

Son adecuados para los clasificadores, componentes, atributos, instancias y operaciones. Una etiqueta {documentación = } se aplica a cualquier elemento y debemos indicar una descripción, explicación o comentario respecto del elemento al que adjuntamos el valor etiquetado.

{ubicación = }: para un clasificador, proporcionamos de cual es parte. Para un componente, indicar el nodo donde se encuentra.

{persistencia = }: puede ir en un atributo, clasificador o instancia. Indica permanencia del estado del elemento al que lo hemos adjuntado. Valores: permanente (el estado persiste cuando la instancia se destruye) y transitorio (el estado se destruye con la instancia).

{semántica = }: especifica el significado de un clasificador o una operación.

{responsabilidad = }: es una obligación de un clasificador.